5-2020

Neusch Automated Filter System: A Custom-Designed Photo-Filter Switching Apparatus for Embry-Riddle's 1 Meter Telescope

Tyler Neusch

NEUSCH AUTOMATED FILTER SYSTEM: A CUSTOM-DESIGNED PHOTO-FILTER SWITCHING APPARATUS FOR EMBRY-RIDDLE'S 1 METER TELESCOPE

By
Tyler Neusch

A Thesis Submitted to the College of Arts and Sciences
Physical Sciences Department
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Engineering Physics

Embry-Riddle Aeronautical University
Daytona Beach, Florida
May 2020

NEUSCH AUTOMATED FILTER SYSTEM:
A CUSTOM-DESIGNED PHOTO-FILTER SWITCHING APPARATUS
FOR EMBRY-RIDDLE'S 1 METER TELESCOPE

By

Tyler Neusch

This thesis was prepared under the direction of the candidate's Thesis Committee Chair, Dr. John Hughes, Associate Chair & Associate Professor of Engineering Physics, Daytona Beach Campus, and has been approved by the Thesis Committee. It was submitted to the Department of Physical Sciences in partial fulfillment of the requirements of the degree of
Master of Science in Engineering Physics

THESIS COMMITTEE:

Dr. John Hughes,
Committee Chair

Theodore von Hippel (Apr 23, 2020)

Dr. Ted Von Hippel,
Committee Chair

Charles Lee (Apr 23, 2020)

Dr. Charles Lee,
Committee Member

Stephen D. Gillam
Stephen D. Gillam (Apr 23, 2020)

Dr. Stephen Gillam,
Committee Member

Dr. Terry Oswalt,
Department Chair,
Physical Sciences

Dr. Karen Gaines,
Dean, College of Arts and Sciences

Dr. Christopher Grant
Associate Provost of Academic Support

Abstract

The one-meter telescope on the roof of Embry-Riddle's College of Arts and Sciences building in Daytona Beach, FL is an advanced piece of technology supporting education and public outreach. One part of this telescopic system is a custom-made Cassegrain Instrument Mount. This one-of-a-kind mount allows for multiple scientific instruments to be simultaneously mounted to the telescope. However, its mechanical layout prevents the use of a traditional device known as a filter wheel.

Filter wheels store multiple photo-filters each capable of filtering out all but certain wavelengths of light for use with a CCD camera. Therefore, a filter wheel allows the telescope user to conveniently select and change between wavelengths detected by a CCD.

During my senior year in Embry-Riddle's Engineering Physics program, two others and myself undertook the task of creating a custom photo-filter switching apparatus that would mount on the Cassegrain Instrument Mount. At the end of this senior design we had a product that had the potential to meet all requirements set for the device, but was not yet technologically mature or operationally stable.

The goal of this thesis was to take the end senior design product and continue its development to a point of completion. Addressed first is a detailed history of the initial conceptual design as well as a description of the prototype (V0) and final senior design version (V1). Once the major problems were identified they were broken down and solved one by one, starting with mechanical modifications and moving on to the overhaul of the controlling software and user interfaces. Results from tests to verify that the system meets all requirements are also described.

Acknowledgements

Above all else I would like to thank every person, family, friend, and otherwise who has supported me on my schooling journey. None of this would be possible without knowing that so many people believe in me. I would also like to thank my thesis advisor, Dr. John Hughes, who has given me endless guidance and advice from the start of this project. Finally I would like to thank Dr. Ted Von Hippel and Dr. Steve Gillam who never failed to give me feedback on how to make my device better and more applicable for the real world.

None of this work would have been possible without the contributions of my BSEP senior design partners Adam Campagnolo and Gustavo Arturo Villarroel.

Contents

List of Figures

List of Tables

Chapter 1

BACKGROUND & HISTORY

1.1 The Problem

On the roof of Embry-Riddle Aeronautical University's College of Arts and Sciences building sits the largest telescope in the southeast U.S. The one-meter Cassegrain telescope is used for education, public outreach, and scientific research and serves students, astronomers, and the broader community.

Figure 1.1a shows a picture of the telescope and fig 1.1b shows a close-up of the Cassegrain Instrument Mount (CIM), a mechanical interface between the telescope optics and the instruments used to detect the collected light.

The CIM includes a slide sized to house a 75mm × 75mm optical filter. Optical filters allow the telescope user to select or limit the wavelengths of light passing through the telescope optics into the detection instrumentation. Inconveniently, and serving as the ultimate motivator for this project, the optical filter must be changed manually, requiring both physical access to the telescope during operation and undesirable time delays.

1.2 The Solution

The proposed solution to this problem was an automated, remotely controlled filter changer that would store a number of filters outside the CIM and insert the user-selected one into the filter slide as needed. The design constraints were:

- Device must be able to handle five different photo-filters
- Must be able to switch between photo-filters in sixty seconds or less
- Must place each photo-filter at exactly the same location within the telescope system
- Must interface with the existing apparatus mount

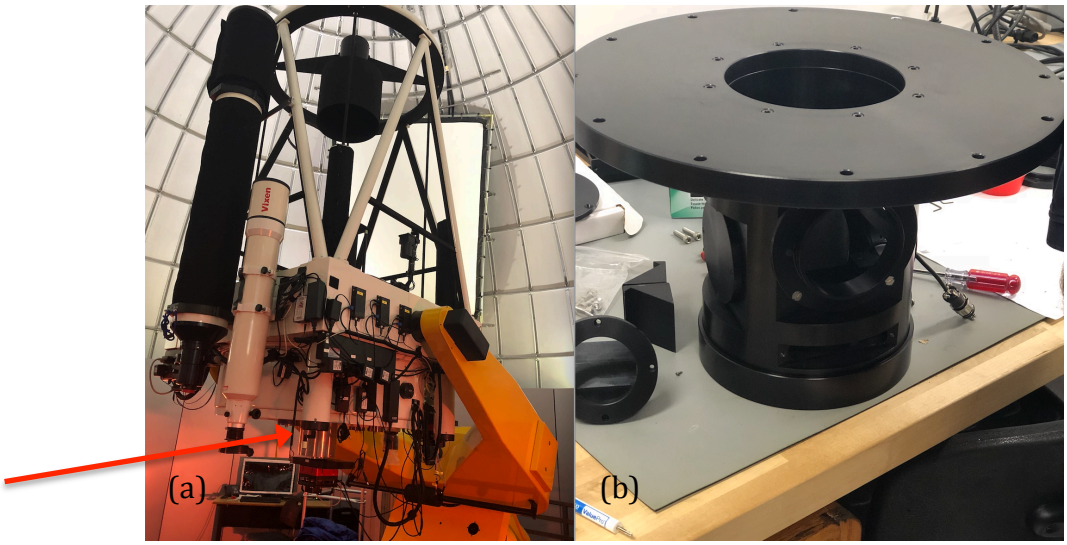

Figure 1.1: (a): Embry-Riddle's 1-meter Cassegrain telescope. (b): Cassegrain Instrument Mount (CIM). The red arrow indicates where the CIM and NAFS will be placed in the telescope system.

The NAFS (Neusch Automated Filter System) was the device created to solve the filter-switching problem and meet all constraints. The NAFS holds all five filters in a vertical stack and uses an elevator-style concept to align and insert the chosen filter inside CIM. Figures 1.2 and 1.3 illustrate the NAFS prototype with its first and fifth filter inserted respectively.

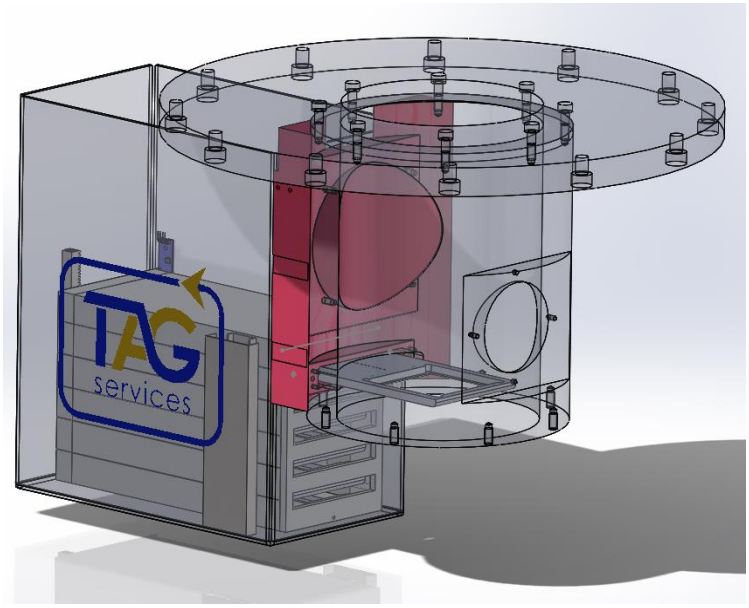

Figure 1.2: CAD drawing of the NAFS prototype device mounted to the CIM with 1st (top) filter inserted. The NAFS is labeled with "TAG services", referencing a BS-Engineering Physics senior design group that developed the initial concept and prototype.

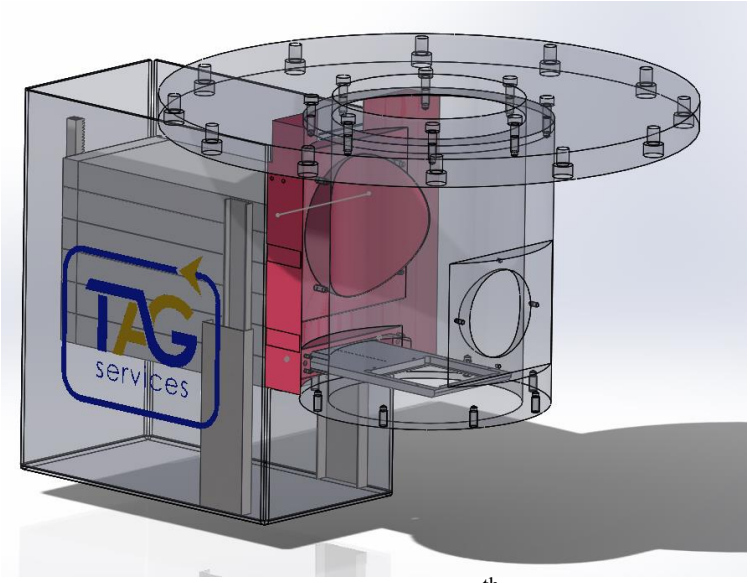

Figure 1.3: CAD drawing of the NAFS device with 5th (bottom) filter inserted. Notice the higher position of the "elevator" within the NAFS as compared to the view in Figure 1.2, aligning the lower 5th filter with the CIM filter slot.

Operationally, the NAFS changes the inserted filter by removing any currently inserted filter, moving vertically to align the chosen filter with the CIM's filter slot, and finally inserting the chosen filter horizontally into the CIM.

1.3 History

This filter changer project was initially proposed and carried out as a BS Engineering Physics senior design project for myself and two other students, Adam Campagnolo and Gustavo Arturo Villarroel. Over the course of two semesters, the proposed device underwent conceptual design, prototyping, and a final build. The result was a technologically immature, semi-working device that showed promise.

1.3.1 Initial Concept

As envisioned by the BSEP senior design team, the initial design concept for the filter changer included two different boxes, one static and one dynamic. The static box, which came to be known as the Outer Housing, would be attached to the CIM and was intended to provide all the necessary structural support for the device. The dynamic box would then mount inside of the Outer Housing and be able to move up and down with the elevator-style motion mentioned above. Figure 1.4 below shows an early sketch which served as the baseline target for our preliminary design work.

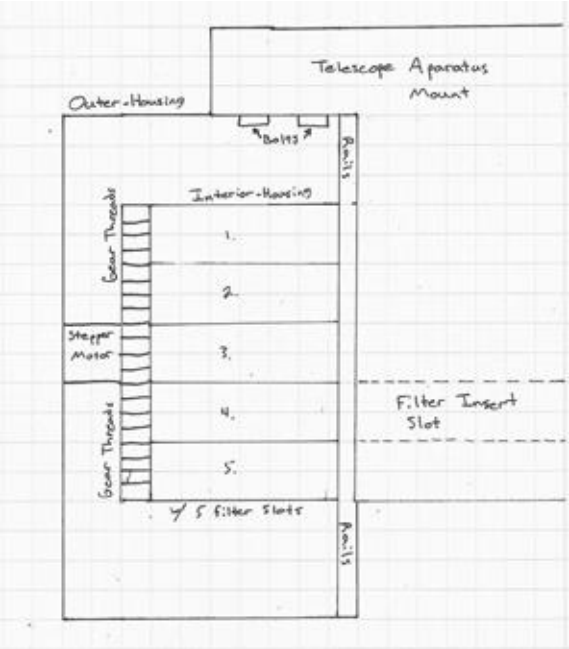

Figure 1.4: Sketch (drawn on 8/30/18) of the filter-changer initial conceptual design.

This sketch was far from a complete picture and left most details unaddressed. Questions such as how to attach the inner stack (dynamic box) to the outer housing in a manner that left a vertical degree of freedom, and how to insert the aligned filter into the telescope apparatus mount were still unanswered.

1.3.2 Prototype (V0)

At this point in development of our senior design project, it was decided that the best way to make progress was to simply start building. Thus the construction of a prototype began. The prototype was never intended to actually mount to the telescope, only to verify the proof-of-concept for horizontal and vertical filter movement. The outer housing was a simple plywood box with no top and only a partial front. The inner stack was also a partially-faced plywood box, roughly half the size in every dimension of the outer housing.

The foremost question after constructing the inner stack/outer house was how to mate them together while providing the required vertical degree of freedom. The answer came in the form of smooth-gliding drawer slides mounted on both sides of the inner stack.

Next to be considered was the mechanism that would horizontally move a desired filter outward from the rest of the device, inserting it into the CIM. After considering a variety of options, the team chose to attach a motor driven push rod that would pass through a hole in the back of the inner stack and push the filter slide out. Each filter slide was mounted to its own linear guide rail within the inner stack to allow for smooth and straight horizontal movement. An extended slide could be retracted using a neodynimium

magnet attached to the push rod. This magnet would attach to a metallic tab on the slide, providing the force required to retract the slide.

The prototype device was controlled and operated using an Arduino UNO and two different sensors. The Arduino was capable of controlling both the horizontal and vertical motors as well as reading values from the two sensors. The first sensor was a photogate that, when triggered, informed the Arduino that the inner stack was lined up vertically in preparation for the insertion of a filter (see figure 1.5 below). The second sensor was an RFID reader that read the RFID tag from the nearest filter slide in order to tell the Arduino which filter was aligned. Figure 1.6 below shows the entire prototype device.

Figure 1.5: Photogate sensor implementation on prototype.

Figure 1.6: Complete prototype

1.3.3 Senior Design Final (V1)

The prototype accomplished all of the goals set for it and succeeded as a proof-of-concept. However, it required many changes to the design and construction before it could be installed on the CIM and telescope. Size and weight reduction, an interface to the CIM itself, Arduino/wire mounting, power source modifications, and a remodeled inner stack were all required.

Size and weight reductions were one of the easiest upgrades to tackle. This is due to the fact that the prototype was designed to have a large amount of extra space. In the next (V1) prototype, the largest dimensions were reduced from 24” × 11” × 11” to 12” × 10.625” × 6”. The material of the outer housing and inner stack were changed to aluminum and 3D-printed plastic respectively. This was done in order to add structural rigidity while also reducing weight.

Figure 1.7: V0 (left) vs. V1 (right) size comparison.

As illustrated in figure 1.7, both the V0 and the V1 have a flat front face. The CIM, to which the filter-changer must mount, is cylindrical, which means that an interfacing adaptor was in need. To deal with the complex features of the CIM, rounded, rectangular and circular slots, (see figure 1.1b) it was decided that a 3D CAD-modeled and printed piece would be best. The CAD drawing for the CIM was acquired and an interface was built around it that would precisely mate the outer housing of our device to the CIM.

Figure 1.7 also shows two white enclosures on the right side of the outer housing. These enclosures were part of the solution to wire management. In V0 the motor drivers, Arduino, breadboard and other wires were all stored in a loose box at the bottom of the device. As this was not acceptable for future versions, V1 included three enclosures built onto the outer housing. The top enclosure visible in figure 1.7 housed the Arduino, the lower one housed the two motor drivers and breadboard for the sensor wiring, and a third, situated on the back face and not visible in figure 1.7, contained both the vertical and horizontal motors as well as the push rod rack and pinion. An added benefit to the enclosures is that maintenance was much easier as all wiring was organized and easily accessible.

With improved wire management, it became feasible to add more sensors into the device. Three sensors where added to aid in position verification (one Hall effect and two micro switches) and three were added to give the ability for a user to control the device without software (one toggle switch and two potentiometers). The Hall effect sensor contributed to identifying the horizontal position of a slide and the micro switches were added to the top and bottom of the outer housing's interior to identify when the inner stack was at its highest and lowest points. The three user control sensors were interfaced to an LCD screen to form what became known as the Manual Control Unit (MCU). Activated by the toggle switch and controlled by the two potentiometer knobs, a user could manually select from various device functions.

Possibly the most dramatic change from V0 to V1 was the design of the inner stack. In both figures 1.6 and 1.7 it can be seen that elements of the inner stack extend outside of the outer housing. Because these protrusions prohibited mounting to the CIM, they were unacceptable and needed to be eliminated in V1. The redesign implemented the necessary changes to completely contain and support the linear guide rails, have a partial front face, and side mounted vertical rails. It also had an extra flange on the back edge to trigger the photogate sensor at the correct position for each filter slide. Figure 1.8 below shows the CAD model created for the 3D print of the inner stack body and includes all of the non-3D-printed elements within it.

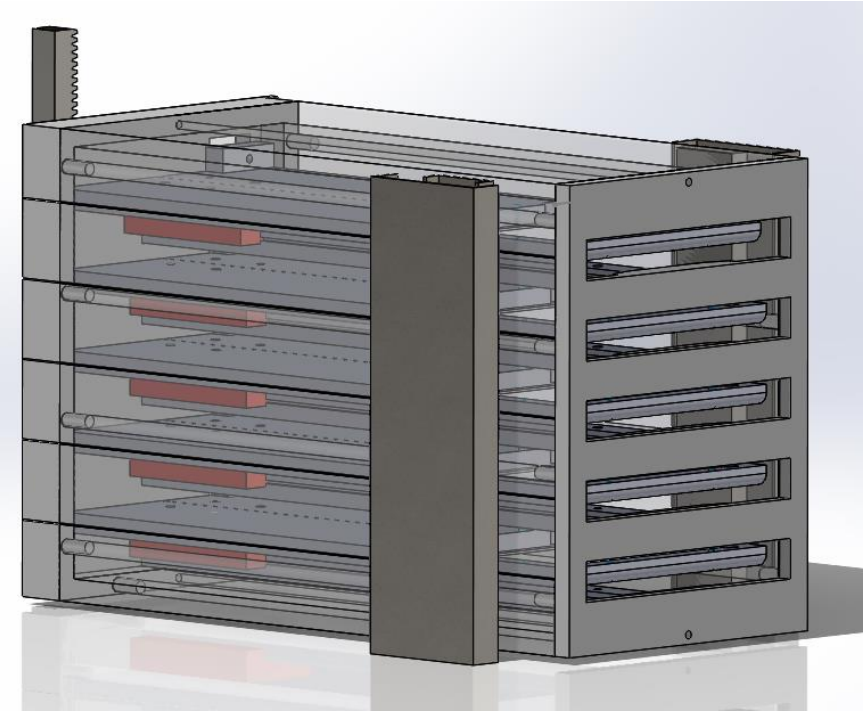

Figure 1.8: CAD model of redesigned inner stack for V1

1.3.4 What Didn't Work

At the end of two semesters of senior design, my team and I were forced to conclude our development on the V1 device. A substantial improvement over V0, the V1 device verified proof-of-concept and was actually able to integrate with the CIM and the telescope system. However, the device lacked the technical maturity and mechanical robustness necessary for implementation and everyday use by the telescope's operators. In other words, the device worked, but only some of the time. There were a few critical issues with the final V1 device and multiple smaller things that required further improvement.

The largest flaw in the V1 was vertical slipping of the inner stack. A motor attached to the outer housing drove the inner stack up or down using a rack and pinion system. (A rack and pinion is basically a gear attached to the motor shaft that mates with a gear track mounted to the inner stack.) Due to flex within the rails that attached the inner and outer housings it was possible for the gear track to move far enough from the gear that the mating connection was lost. This would cause the inner stack to drop unexpectedly from an elevated position to an unspecified lower position. Furthermore, if a filter slide was extended at the time of this malfunction, the weight of the inner stack would be placed purely on non-load bearing elements.

Another major problem was that the pushing mechanism would routinely fail, causing the desired filter slide to either be partially inserted or not inserted at all. Although this at first appeared to be a hardware-related failure, its cause was later

isolated to a coding error. In order for a slide to be inserted completely into the telescope system the inner stack must be accurately placed within the outer housing. This is because the push rod needs to be lined up with the hole for the desired filter slide on the inner housing. If it is not aligned mechanical interferences negatively affect the performance of the push rod.

The Hall effect sensor, having full responsibility for horizontal position verification, worked less than 20% of the time. If a magnetic field passes around the Hall effect sensor a voltage is supplied to the Arduino. With this in mind micro-neodymium magnets were placed on the bottom of each filter slide in such a way that when fully extended the slide's magnet would trigger the Hall effect sensor. Thus, the proper horizontal position of an inserted filter can be verified by ensuring the Hall effect sensor was triggered. Two things caused the persistent failure of the sensor-magnet system: improper positioning of the magnet and a polarity discrepancy. For proper operation, precise alignment is required between the magnets and the Hall effect sensor. Achieving this alignment is difficult and the V1 device did not do it sufficiently well.

While the problems discussed above needed to be solved in order to achieve a fully working device, there were also problems that needed to be solved purely to make integration to the pre-existing telescope system possible. The largest issue being a control interface that improved on V1's simple but not user-friendly command prompt setup. The V1 was controlled by responding to text-based command prompts interfaced through a COM port terminal. Furthermore, the V1 software had to provision to utilize the manual control hardware built onto the outer housing. Clearly, a proper user-interface was required.

1.3.5 Summary of Prior Work and Goal

The sections above describe the work performed during my BSEP senior design. The goal of this MSEP thesis is to finish development of this filter-changing device by addressing all shortcomings described above, characterizing its performance and testing it to ensure all design requirements are met, as well as ensuring its readiness for installation and use on Embry-Riddle's 1-meter telescope. The end product, now known as the NAFS V2, and the steps undertaken to obtain it are presented in this thesis. While laboratory closures due to COVID-19 have prevented full characterization of NAFS V2 performance, I completed and report on a set of tests to verify the device meets all design requirements including all requests from the faculty astronomers.

Chapter 2

Mechanical Modifications

2.1 Overview

Figure 2.1 shows mechanical overviews of the final NAFS V2.

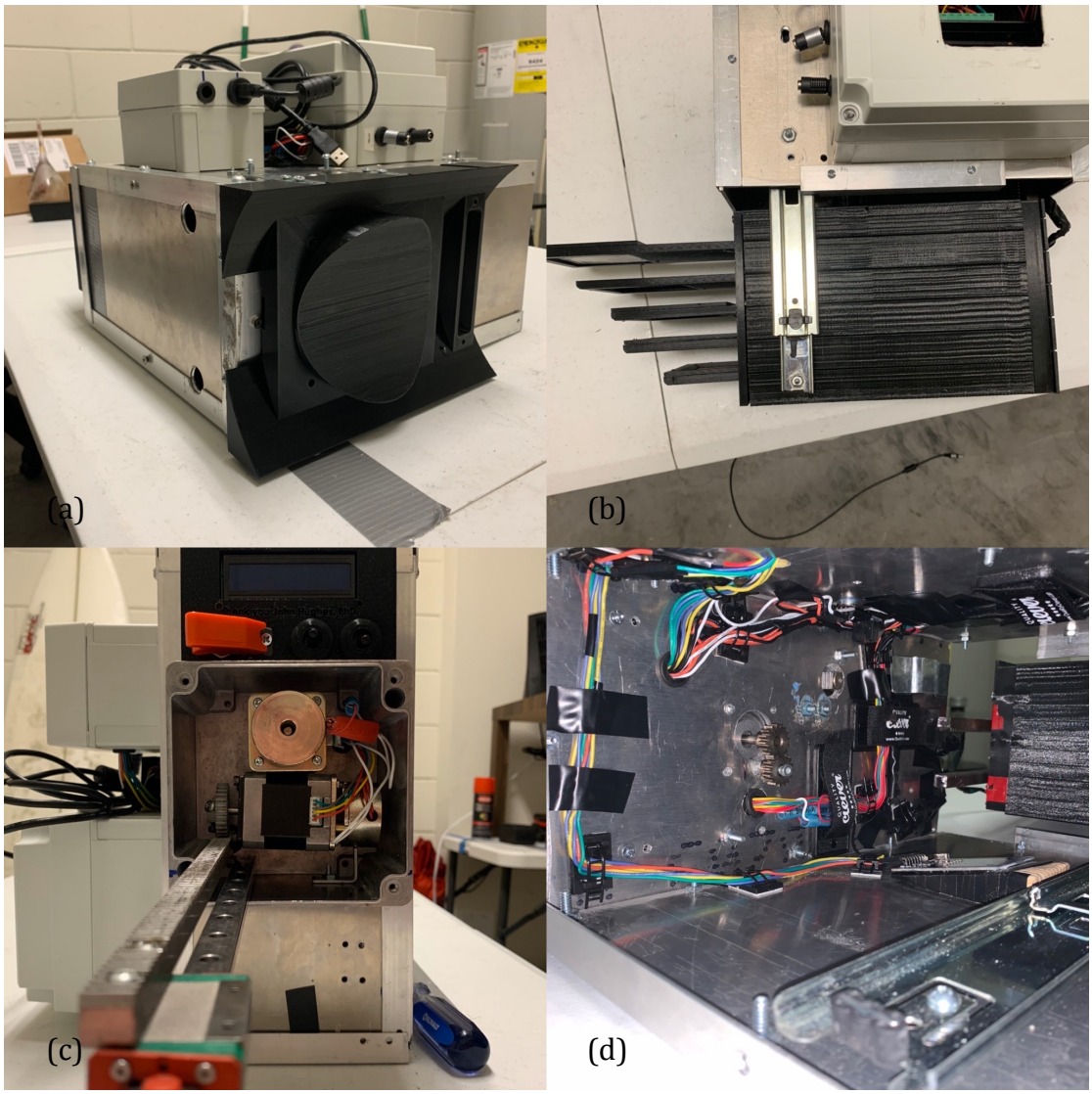

Figure 2.1: Mechanical Overview of NAFS V2 subsystems. (a): Outer-Housing, (b): Inner Stack, (c): Motors, (d) Sensors

This V2 filter-changer consists of four subsystems: the Outer Housing, the Inner Stack, Motors, and Sensors. The Outer Housing is an aluminum casing responsible for supporting all of the system's elements. Also included in the outer housing are the vertical rails and CIM interface (black feature in figure 2.1a). The Inner Stack is the body that moves vertically within the outer housing and contains all the photo-filters. It is comprised of the 3D-printed casing, five linear guide rails, and five filter slides. The Motors subsystem includes both the vertical and horizontal motors as well as the previously described push-rod mechanism and vertical rack & pinion system. The rack and pinion system is visible in figure 2.1d instead of in figure 2.1c with the rest of the Motors subsystem. The Sensors group includes the RFID, Hall effect, photogate, and microswitch sensors. The photogate and RFID sensors as well as wire management associated with every sensor is visible in figure 2.1d. At least one major modification was needed within three of the subsystems to meet all requirements and goals set for the V2. The inner stack required no major modifications but was moderately impacted by almost every change accomplished in the other subsystems.

NAFS V2 was designed and built to meet the following requirements in addition to those already met by V0 and V1:

- Stabilize the inner stack
- Eliminate all slide extension clearance issues
- Ensure Hall effect sensor reads every slide
- Be able to manually retract an inserted slide in case of system failure

2.2 Subsystem Mods: Motors

In the NAFS V1 the issue most likely to cause a total system failure was the inner stack becoming unstabalized at certain points, which allowed undesired freedom of movement. For this reason, stabilizing the inner stack was made a top priority and a required modification for the V2. This would have a direct impact on the rack and pinion portion of the Motors subsystem.

In the engineering world, it is often desirable or required to identify the exact cause of a system failure. Among other benefits, this identification can help avoid similar or repeated failures. Sometimes, and more often in custom, one-of-a-kind applications, it is sufficient and more efficient to focus on a particular solution rather fully diagnosing the cause. This was one of those times.

The problem, as expressed in paragraph 2 of the "What Didn't Work" section in Chapter 1, was due to the rack and pinion system that controls the inner stack's vertical motion becoming unmeshed. For some reason the inner stack was able to move very minutely in the horizontal direction which would allow the rack to move away from the gear. To determine the cause of this horizontal motion the inner stack was cycled up and down while I applied a force in the horizontal direction. During this test it was seen that at higher vertical positions the inner stack was more susceptible to horizontal motion. I hypothesized that, for undetermined reasons, the inner stack itself was moving at slight

angle off of vertical. Possibly it was due to unequal flex within the vertical rails, an unbalanced load on the inner stack, or an unseen force/torque placed on the stack. Instead of further testing to determine the exact reason I decided to implement a solution using another observation from my first test.

I had noticed that due to the aforementioned flex within the vertical rails, the inner stack could be forced to move at a zero angle and thus never lose the rack and pinion mesh. The problem could therefore be solved by providing the force required to keep the rack pressed into the gear. It was determined that this could be accomplished by adding a second gear and rack to the rack and pinion system. The second gear would be a free spinning (idler) gear that meshed with the primary gear and the second rack. The second rack would also be oriented in the opposite direction of the primary one so as to create horizontal force in the correct direction. A depiction of the "double rack" system is given in figure 2.2 below.

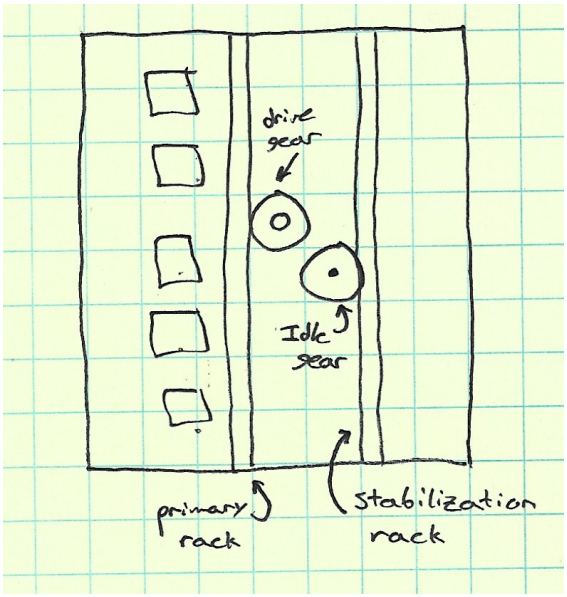

Figure 2.2: Double rack system. As implemented, the idler gear is engaged by the drive gear while meshing with the stabilization rack.

The most difficult part of implementing the double rack was the spacing with preexisting elements. The primary gear was in a fixed position and unmovable. The primary rack was initially on the right hand side of the drive gear shown in figure 2.2, however, due to the fixed push rod holes in the inner stack there would not be enough room for the idle gear and the second rack on the left. So the primary rack was removed and remounted on the left allowing the extension pieces to go on the right.

Unfortunately, the right hand side also did not have unlimited real estate. Mounting hardware on the outer housing meant that the idle gear did not have enough space to be placed directly in line with the primary gear. So I determined the maximum allowed right offset from the center of the primary gear and created a triangle with a hypotenuse of two times the radius of the gears. Running these values through a simple Pythagorean's theorem yielded the vertical offset required for the idle gear to fit. Using

the horizontal and vertical offsets in conjunction with the primary gear's position I was able to mount the idle gear with a perfect mesh of the primary.

After implementation of the double rack the inner stack was run through the same test as before with an extraneous horizontal force at various vertical positions. The test proved that the inner stack no longer lost the rack and pinion mesh and therefore all vertical slipping was mitigated. Furthermore, by placing a level on top of the inner stack and moving it through every vertical position, I verified that a zero angle was now constant.

2.3 Subsystem Mods: Outer Housing

2.3.1 Reload Rails

While not critical to everyday operation, an easy way to access and change the photo-filters without a full system unmount was a top request from the faculty astronomers. The only way to accomplish this was to have the inner stack extend entirely out of the outer housing to a spot that it and its contents could be accessed. The easiest way to accomplish this was to replace the current vertical rails with ones capable of extending the extra distance. Since the current rails were at full compression when the inner stack was at its lowest point and full extension at the highest point, the extra distance needed would simply be the height of the inner stack itself (5.75") plus an extra inch for precaution.

I immediately identified two potential problems with the rail extensions. First was that rails capable of extending the extra distance might not fit within the outer housing. This would mean the rails might protrude out of the top/bottom of the outer housing, creating mechanical interferences. Purchasing and installing rails of the required length confirmed they were in fact too long to fit within the outer housing. They were made to fit by removing some extraneous length from one end of the rails and by drilling new mounting holes.

The other potential problem was that in the V1 system, the rails extended as the inner stack moved upwards. This meant that with the extended rails the inner stack would be pushed up and away from the top of the outer housing when access was required. This however was impossible due to the fact the CIM partially extends over the top of the outer housing. Thus the inner stack's required movement would be impeded by the overlying telescope. The obvious solution to this was to make the floor of the outer housing removable and send the inner stack through the bottom instead. This was accomplished by inverting the rails 180 degrees. This made it so that the inner stack's highest position was at the full rail compression the rails extended as the system moved downwards. Figure 2.3 shows the outer housing with the inner stack dropped to its lowest position, allowing manual access to the filter slides.

The result of these rail modifications was a successful method of easy access for the inner stack and all of the slides it holds. It was implemented into the user interface so that when selected the user would be told to remove the floor of the outer housing before continuing. The motors would then push the inner stack down or pull it up depending on which reload function was selected.

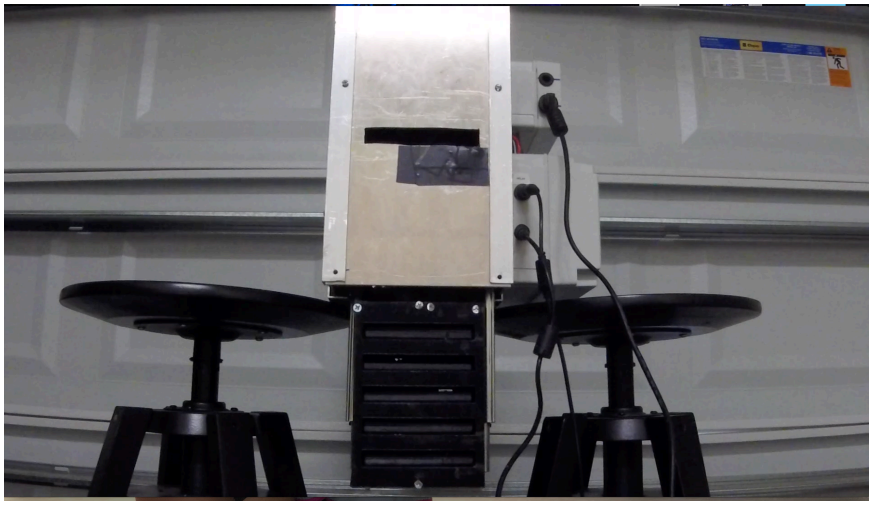

Figure 2.3: The V2 outer housing and inner stack dropped into its manual access position.

2.3.2 System Failure Slide Retrieval

During the design process the faculty astronomers requested many precautionary, but understandable, features. One of these requests was for a way to retrieve an inserted slide should the system encounter an error or failure. Once a slide is pushed into the CIM, it is held in place by the locked push rod. Should there be a loss in power, a motor malfunction, or if the retraction mechanism should fail, the slide could get stuck inside the telescope. This would create the problem of the entire NAFS V2 device being unmountable from the telescope without potentially damaging an expensive photo-filter.

I identified two things that would need to happen to recover a potential stuck slide. First the push rod would have to be unlocked and/or retracted. Second, there would need to be an access port somewhere that would allow a user to manually retract the slide. To deal with the push rod I implemented two custom functions into the user interface. One just turns the pusher motor's brakes off and the other retracts only the push rod itself. For the access port I determined its best location to be on the side of the 3D printed CIM interface. I placed a 5/8" diameter hole directly in line with the slide slot on the interface. This makes it so that at any level of extension I can use a needle nosed tool to reach in and retract the slide manually.

2.3.3 Updated Mounting Procedures

Once completed and ready to be put into operation, the NAFS V2 will be mounted onto the CIM. This mounting requires particular bolts of a specific length attached to the CIM. Due to a miscommunication, the provided bolts were too short and replacing them with longer ones required an unacceptable disassembly of the CIM. Thus a rapid redesign of the mounting procedures and hardware began.

My initial plan called for two 3" bolt studs to be dropped in from the top of the CIM. The provided bolts were, instead, two ¼" studs. The first obvious step was then to

extend the studs to the necessary length. This was accomplished with the use of a coupling nut (the blue element in figures 2.4 & 2.5) and another stud that would make up the rest of the length.

At this point an impasse was discovered. The original mounting procedures implemented toggle bolts that could only be unscrewed from the top. However, the top of the screws would not be accessible without removing the CIM, which was now not possible. So it was determined that the new mounting hardware had to be removable from below without impacting the CIM. Typically a simple nut and washer would allow for this, but due to the 1.5" diameter hole intended for the toggle bolt and the fact that the coupling nut extended into this hole things would need to be more complicated. First of all a washer (top red element in figures 2.5 and 2.6) with an inner diameter slightly larger than the coupling nut and an outer diameter larger than 1.5" was used to plug the hole in the roof of the outer housing. Next, a stainless steel thick spacer (green element in figures 2.5 and 2.6), whose inner diameter was also larger than the coupling nut, was slid over the coupling nut and extension bolt. Finally a nut and washer (lower blue and red elements in figures 2.5 and 2.6) were placed on the extension bolt to hold everything in place. The AutoCAD model in figures 2.5 and 2.6 represents this hardware system. Note that the grey plate in the figures 2.4 through 2.6 represents a small section of the outer housing roof and that there are two of these hardware systems so that both holes in the roof are filled.

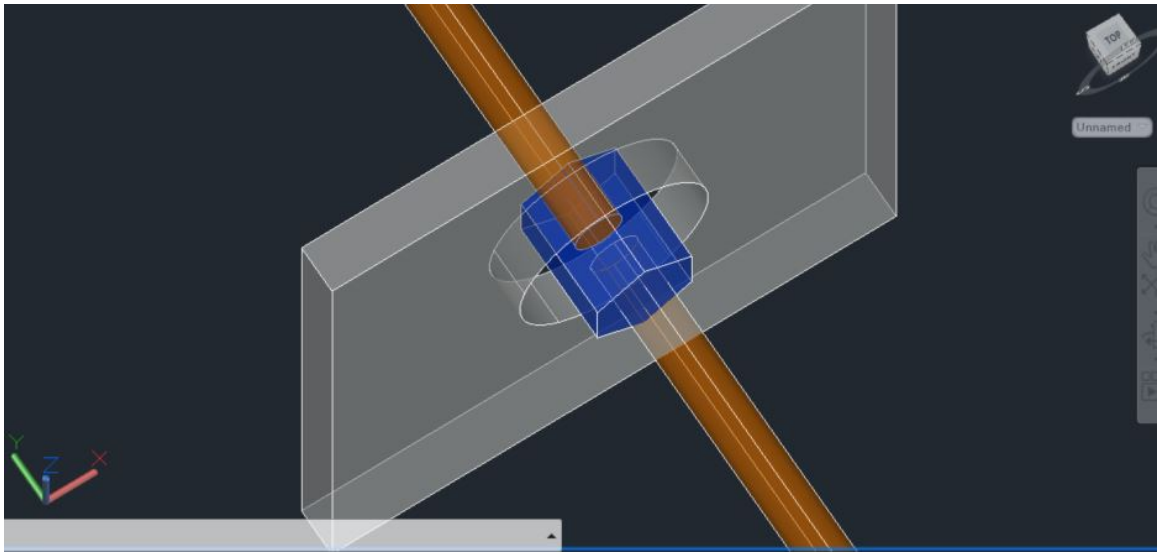

Figure 2.4: Coupling nut with extension bolt

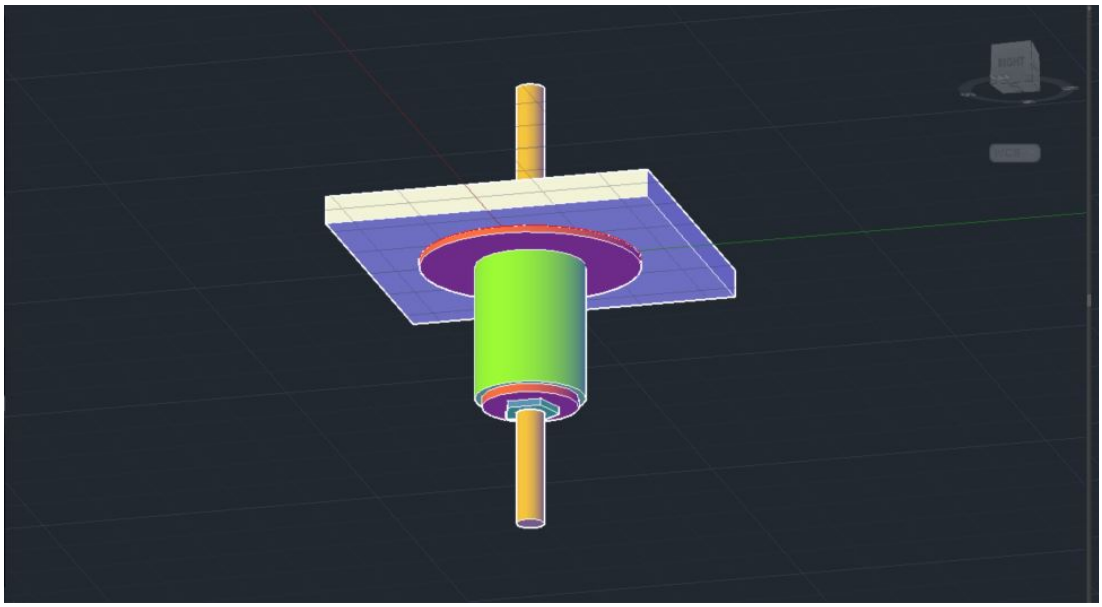

Figure 2.5: Updated hardware mounting system

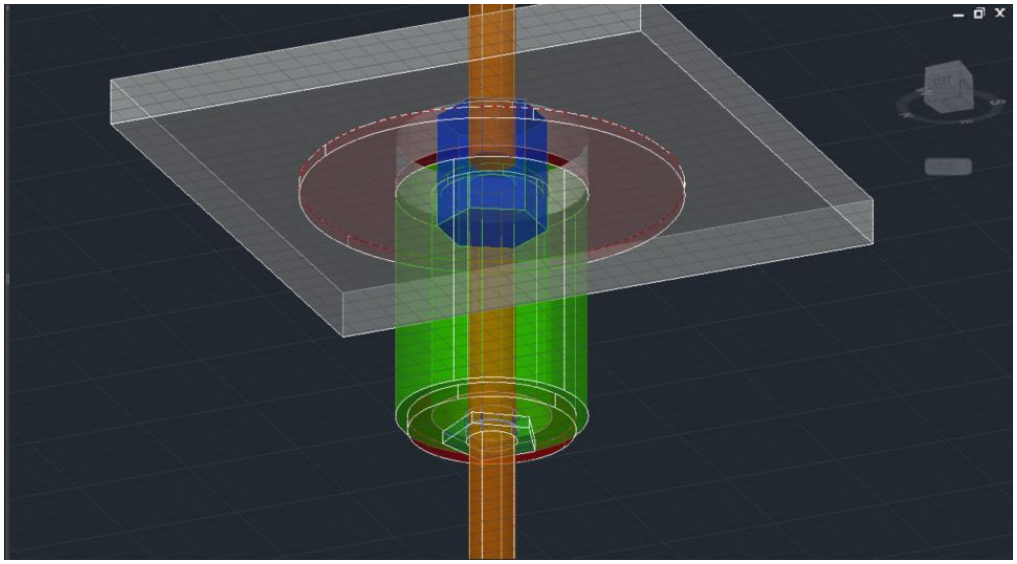

Figure 2.6: X-Ray view of hardware mounting system

2.4 Subsystem Mods: Sensors

2.4.1 Hall Effect Polarity

During the final days of upgrading from the V1 to the V2 system, one problem continued to persist. The Hall effect sensor only worked some of the times. I knew from the start

that the spatial range of this sensor was very limited, therefore the magnets on the slide being passed over it needed to be in a very specific location. At the time it was reasonable to believe that if a slide wasn't detected by the Hall effect sensor it was because its magnet was attached too far from the required position. To fix this I found one slide that worked reliably and measured the exact position of its magnet relative to the edges. I then re-epoxied the magnets for every other slide in that same position. Surprisingly, this still did not mitigate the issue entirely. I now had three of the five slides work every time while the other two never worked.

Since the position of the slide magnets was now consistent across slides, I eliminated that as the source of failure. Instead I focused on the sensor itself. I removed, replaced, and adjusted the Hall effect sensor to try a multitude of variants. At one point, it was secured upside down. When testing this variant I observed that the two slides that had never worked before were now functioning perfectly. Furthermore, the three that had always worked before were no longer giving any readings. The fact that the relative orientation of the magnets to the sensor had such a clear impact could only mean that I had a polarity issue. Upon checking the polarity of the magnets my suspicions were confirmed. The three magnets that had always worked had their positive side facing down, while the other two had the negative side downwards. By orienting all five to have their positive side down I was finally able to achieve a good Hall effect reading on all slides.

2.5 Subsystem Mods: Inner Stack

The inner stack was the only subsystem that did not require significant modification from V1 to V2. The only issues it did have were mitigated during the major modifications of other subsystems.

When defining the V2 upgrade requirements I identified the mitigation of all slide extension interferences as a top priority. Initially this seemed that the root of this problem was some sort of mechanical interference within the inner stack. However, upon further investigation this was actually caused by the misalignment of filter slides with the slot on the front face of the outer housing. Since this was now an alignment issue I was able to resolve it simply by telling the motors to move up or down a few extra steps to achieve proper alignment. This process is discussed in more detail in part 3.

2.6 Summary

Through the work discussed above, all mechanical shortfalls identified in V1 were successfully mitigated. Vertical slipping was remedied by stabilizing the inner stack with a second rack and pinion. An access port was placed in the front interface mount to recover an inserted slide in case of a system failure. The position and polarity of all slide magnets were made uniform and as a result the Hall effect sensor is able to consistently verify horizontal position. Longer vertical rails were inverted and mounted in order to

allow for maintenance of the inner stack and the exchanging of filters in the system without requiring an unmount of the V2 from the CIM. New mounting procedures were also addressed to allow the V2 to be mounted/unmounted without the need for removal of the CIM. All together these changes yielded a mechanically stable system that was structurally ready for installation on Embry-Riddle's 1-meter telescope.

Chapter 3

Software Modifications

3.1 Purpose/Overview

The software flashed onto the Arduino for the V1 version of the device had limited functionality, lacked a user-friendly interface, and was not intended for final implementation. It lacked robustness, error checking and reporting, had many failure points, and provided little flexibility to the operator. An almost complete overhaul was required for the operational V2.

One of the main upgrade requirements identified for V2 was a Graphical User Interface (GUI). The GUI would provide a user, who may or may not be familiar with the system, a screen of buttons and text boxes with everything necessary for operation of the device. In order to make the GUI a reality the following three things were required:

- The Arduino's software (called SoC) needed to be able accept commands from and send information to a non-Arduino software
- A separate software that provided the underlying code for the GUI and was capable of running on any PC needed to be created
- Continuous two-way communication between the SoC and GUI needed to be established

3.2 SoC Main Loop Restructuring

When reviewing the V1 code in preparation for the V2 modifications I found that the flow of the main loop was following an unacceptably restricted flow of logic. A user had to first run the initialization sequence, then had to select a slide, then had to retract said slide. On top of that, separate user inputs were required for each step. I determined that by stripping the main loop and functionalizing everything the code could be optimized while also allowing for a freeform path of commands. This would allow a user to perform individual operations without being limited to the previous strict sequence.

Modifying the code started by creating functions for every operation. Non-logic based operations such as reading a sensor or moving a motor were turned into functions. Similarly, all top-level operations such as slide selection, initialization, and reloading were reimplemented as functions. The main loop was then written to be a large switch statement. A user enters a single command associated with the desired operation, triggering the associated switch statement cases and calling the top-level function for that

operation. The list of functions and switch statement cases are provided below.

Table 3.1

Functions List	
Function Name	**Description**
Pin_ISR	Reads the toggle sensor on the MCU
UserInput	Gets a user inputted command from the serial buffer
PhotoGate_Stack	Reads the PhotoGate sensor
MoveStack	Moves the vertical motor a specified vector
MoveStack_1	Moves the vertical motor a specified vector with greater speed than “MoveStack”
RFID	Reads the RFID sensor
MovePusher	Moves the horizontal motor a specified vector
Manual_Selection	Reads the potentiometer sensor from the MCU
Hall_effect	Reads the Hall effect sensor
Initialize	Runs the initialization sequence
Reload	Runs the reload sequence a specified direction
Slide_Select	Runs the slide selection sequence to insert a new desired photofilter
Sensors	Sends sensor data to GUI
Sensors_f	Sends sensor data to GUI with terminating character
man_sel_task	Runs a desired non slide select task from MCU

Table 3.2

Switch Statement Cases	
Case	**Operation**
n	Display sensor data
e	Run the initialization sequence
u	Reload Down
s	Reload Up
c	Extend Pusher
h	Retract Pusher
t	All motor brakes off
A	Slide 1 select

B	Slide 2 select
D	Slide 3 select
F	Slide 4 select
G	Slide 5 select
X	All brakes on
Z	Clear error

As evidenced by the operations performed by the different switch cases, the user has many more options and much more control than in V1. Additionally, the switch statement directly correlates with the GUI described in the following section (essentially, each of the switch cases represents a button that the user can hit in the GUI).

3.3 The GUI

Upon initial concept it was thought that the graphical user interface would simply be the interface a user would use to pass commends to the V2. However, it became evident as the interface evolved that it also needed to allow the V2 to communicate with the user. Information on current status, sensor readings, error encounters and more all needed to be relayed to the user. Fortunately, it was straightforward to implement such functionality using Visual Studio and its C# language. As is typical, the GUI started with simple, limited functionality and expanded as required.

I identified the following four main functions as operational requirements for the GUI:

- A way to connect to the system
- A way for the user to select a desired slide
- A way to view the current sensor readings from the system
- A way for the user to invoke special operations

The method for connecting to the system will be discussed in more detail in the "Two-Way Communications" section, but essentially included a drop down box to select a Serial COM port and a connect/disconnect button.

To select a slide or invoke a special operation, buttons were created for each option and then grouped together. Selecting one of the buttons would be in principle just like jumping into the switch case in the SoC code associated with that option. For example if the "Slide 3" button is hit, the SoC code jumps into case "D" and runs the code necessary to insert slide 3. For the requirement of viewing the systems sensor readings, a group of text boxes were created that display the sensor data received from the SoC.

Once the above four requirements were met, I worked closely with faculty astronomers to determine their further wants and needs for the interface. They requested:

- Current system status (moving/stationary, current horizontal/vertical position)
- A way to denote if the system encountered an error and what that error was

- An event log

The system status requirement was accomplished by a series of indicator lights and progress bar sliders. One indicator light changed red and labeled itself as “Moving” if any motor in the system was in motion; otherwise it was green and labeled “Stationary”. Another indicator light was for the insertion status/horizontal position of any slide. If inserted the label changed to “Inserted” and turned red, if retracted the label became “Retracted” and turned grey. The vertical position status was tackled a little differently than an indicator light. Whereas both the horizontal status and motion indicators had Boolean values (moving/stationary, inserted/retracted) the vertical position could be expressed as a value between 1 and 100. A value of 1 indicated the system was at its lowest vertical position and 100 representing its highest. This value could then be assigned to a progress bar so a user could tell the vertical position with a quick glance.

To implement a way to track errors the framework needed to be worked into the SoC code, which will be discussed in the “Other Noteworthy Upgrades” section. The error-tracking group built into the GUI had three elements, an indicator light that turns red when an error is encountered, a text box that lists the error code for the error, and a reset button that is hit when the error has been mitigated. If an error is encountered a message box will appear describing the error, the most common cause, and how to fix it. All operation buttons on the GUI will become disabled until the error is cleared. The list of error codes is given in table 3 below.

Table 3.3

Error Codes	
Error #	**Description**
1	Filter not completely removed
2	Filter not completely inserted
3	Incorrect filter vertically aligned
4	Homing sequence failed

The event log is a large text box that, like all other text boxes in the GUI, is locked to the user and can only be updated by the underlying GUI code. In the underlying code for each button and user selectable feature I added the code necessary to add an entry to the event log text box upon user interaction. In other words, if a button is hit, a new line appears in the event log describing which button was hit and what the system is doing in response. A multitude of other things such as system status changes and error encounters are also documented in the event log. Every event in the log is time/date stamped in case a user needs to retrospectively review the systems usage or movement. The log also has a save option that will save the log in its entirety to the PC’s “TEMP” directory with a time stamped name. An example of the event log displaying an error and being saved is shown in figure 3.1.

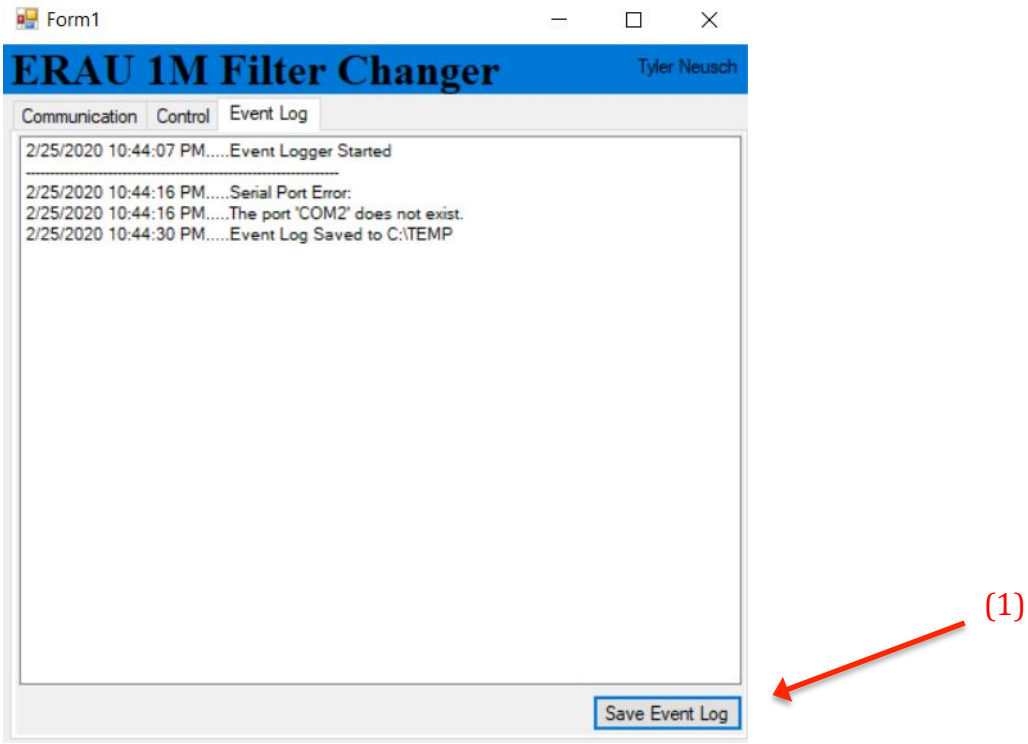

Figure 3.1: Event log example

The creation of the event log prompted me to move the GUI towards a tabbed design. The main tab contained everything mentioned thus far spare the connection group, which was given its own tab, and then a third tab was created to contain the event log. The tabs and features of each are given in figures 3.1,3.2, and 3.3 as well as table 4.

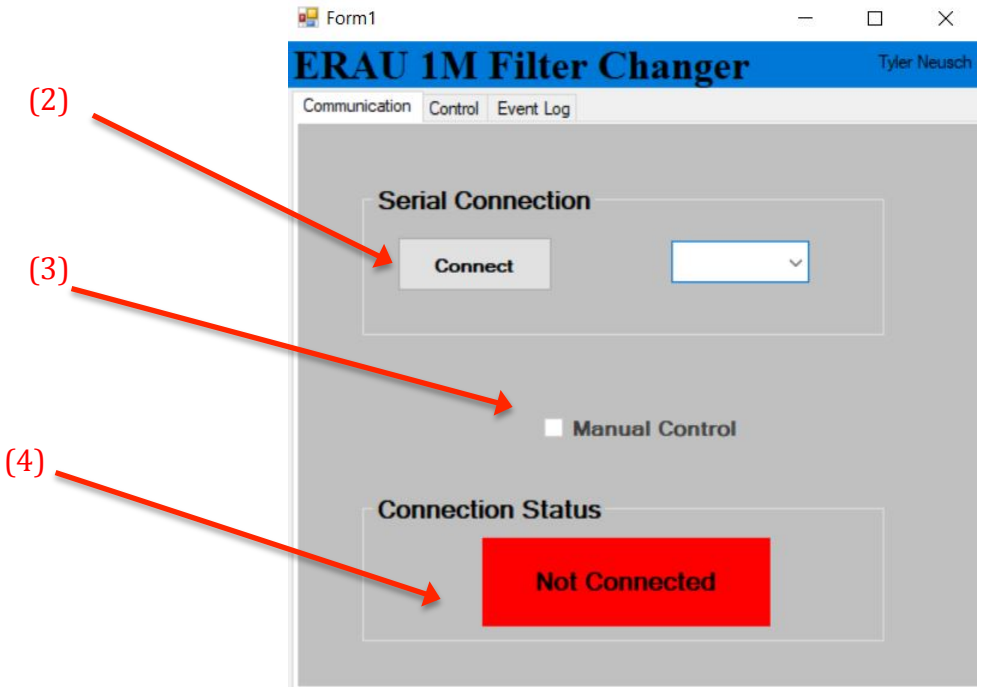

Figure 3.2: GUI communication tab

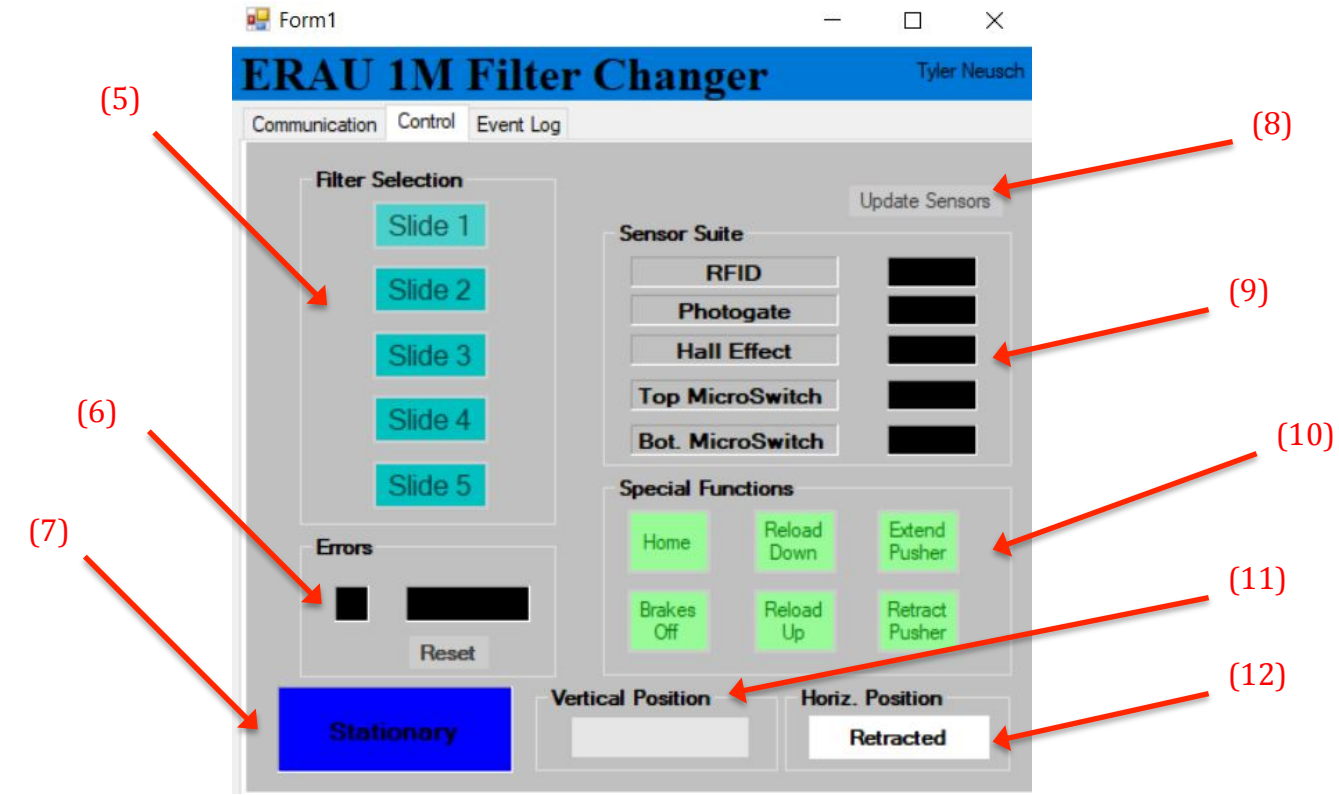

Figure 3.3: GUI main control tab

Table 3.4

GUI Features List	
Feature #	**Description**
1	Save button for Event Log
2	Serial Communication initialization group: Drop down box selects COM port. Connect/Disconnect button starts/ends the connection
3	Checkbox to indicate MCU has been activated. Checking this will allow sensor updates to continue throughout manual control
4	Serial communication connection status
5	Slide selection buttons: An active slide will appear as a different color
6	Error group: Square is error indicator light. Rectangle displays the error code. Reset button clears the error to allow standard operations to continue.
7	System motion indicator light
8	Update sensors button: Requests current sensor data from SoC without a movement operation
9	Sensor Suite: Displays system sensor data
10	Special Functions buttons: 6 different operations found useful during maintenance or error mitigation
11	Vertical Position progress bar: Shows vertical position (0-100) of the inner stack
12	Horizontal Position indicator: Lists whether the push rod is extended or retracted

3.4 Two Way Communications

All communications to and from the SoC to the GUI happen via a Universal Serial Bus (USB) connection. The USB cable plugs from a serial port on the GUI PC directly into the Arduino. The first step to communicating between the two pieces of software is to initialize the connection. This is where the “Serial Connection” group in the GUI comes in to play. Once the User selects the COM port that the USB cable is plugged into, hitting

the “Connect” button runs through the command required for serial initialization. Meanwhile on the SoC, the code is looping continuously over the switch statement until serial traffic is present. Once there is some serial data for the SoC to read, it grabs the line of data and steps into the rest of the code.

How does the SoC know whether or not the serial traffic is actually from the GUI? How are multiple pieces of data sent and interpreted at once? How does the SoC know it has received all the GUI’s data? The answer to all of these questions lie within a standardized message structure. By uniformly using certain characters to denote commands, the start/end of a data set and separate pieces of data from each other within the set both sets of software know what is what. A table of these message characters is given in table 4 below.

Table 3.5

Serial Message Special Characters	
Character	**Meaning**
*	Start of sensor data set
^	Separates data pieces within the data set
!	End of sensor data set
$	End of serial traffic for the requested operation
#	The next character is the case for the switch statement relating to the desired operation

Messages from the GUI to the SoC are very simple. The ‘#’ character is sent first to tell the SoC a command is coming and is followed with a character from table 2 (Note that upon sending this message the GUI is locked from further user input until the requested operation has finished). Messages going the other way though are much more complex since there is a lot more data being sent. These are the messages that contain sensor readings, system status, and error codes. The “sensors” function grabs all the necessary pieces of data, packages them together into a single line string along with some of the characters from table 5 and prints the string to the serial buffer. The order the data is organized into is listed below.

- RFID
- PhotoGate
- Hall Effect
- Top Micro-Switch
- Bottom Micro-Switch
- Vertical Position
- Horizontal Position
- Error Code

The sensors function is called at predetermined points throughout all of the operations a

user can invoke. At the end of the operation the “sensors_f” function is called which does the exact same thing but includes a ‘$’ character at the end of the string. The reasoning for this extra character is described below. An example of the string created by and sent from the “sensors_f” function would be as follows.

***3^1049^1000^0^0^50^0^3!$**

Just as the data needs to be packaged by the SoC prior to sending, it needs to be unpackaged and separated back into its elements by the GUI. This is done by a reverse sensors function that looks for the special characters and saves the values between them. Just as the SoC waits for the ‘#’ character, the GUI waits for the ‘*’. Once it has found an asterisk it saves each character until it gets to the ‘^’. After it reads the caret character it knows it has obtained all the characters associated with the RFID sensor value. That value is then assigned to its designated text box on the main GUI screen. The code then moves on saving characters until it gets to the next caret and so on and so forth. This goes on until the ‘!’ is encountered. At this point the GUI software knows it has obtained the entire data set and proceeds to check one final character. If the ‘$’ is the following character then the transmission from the SoC is over and the GUI switches back to user control. If a ‘$’ is not the next character then the SoC is still in the middle of an operation and thus still sending updated data messages. In this scenario the GUI software is looped back to the start of the reverse sensors function and waits for the next ‘*’ to repeat the process again.

After the requested operation has finished and the SoC has sent all of the data, it loops right back to waiting for more serial traffic. This technique of doing one operation at a time and continuously looping instantaneously while waiting for a request became very efficient and opened the door for implementing something that eluded the senior design team; the Manual Control Unit.

If by some means the SoC or the GUI misses a start special character the ensuing serial traffic will be discarded. While this would delay the execution of a command it would not cause a system failure. The system would continue looking for that start character so by simply sending the request again normal operation would resume. There is the potential for the GUI to become stuck in the “waiting for data” loop if a special character is missed. However, either restarting the serial connection or the GUI itself could remedy this.

If use of the GUI is not possible for certain applications, but full use of two way communications is desired, any command line script or code capable of writing serial data to a COM port can accomplish this. This method was used to great effect during development of the V2. By writing serially to a COM port a “#”, followed by a character from table 3.2 corresponding to the desired operation, any operation can be executed. In turn you will receive back the raw data set message from the SoC.

3.5 Other Noteworthy Upgrades

The Manual Control Unit (MCU) was the group of sensors and an LCD screen built onto the outer housing that would offer a user the ability to control the system without a GUI or separate PC. The idea was that, when a toggle switch was thrown, the SoC would begin reading the potentiometer selection knob and perform the operation associated with that value. The problem came from the fact that during V1 development we were unable to code the MCU toggle as an interrupt. With the new SoC software structure developed for V2 however, an interrupt was not needed. Since the software continues to loop instantaneously while waiting for serial traffic I simply added a check of the MCU toggle to the loop. So now the loop checks the toggle, checks for serial traffic and repeats until one of them return a non-zero. Even if serial communication has already been initialized with the GUI the MCU can still activate as long as an operation is not in progress and/or data is being sent. Table 6 shows the different operations that are selectable with the MCU and the associated potentiometer value.

Table 3.6

MCU Operations	
Potentiometer Value	**Operation**
0-97	Resting Mode
97-193	All Brakes ON
193-289	Vertical Brakes OFF
289-385	Pusher Brakes OFF
385-512	Slide 5
512-639	Slide 4
639-766	Slide 3
766-893	Slide 2
893-1023	Slide 1

Figure 3.4: Manual Control Mount activation/selection

Error checking was another major advancement from V1 SoC software to V2. In V1 there was little to no error checking and as a result the system could move in a manner that proved harmful to itself. The aforementioned error codes displayed by the GUI and listed in table 3 play an even bigger part in the SoC software. I created a global error variable that is checked prior to the motion of any motor. If the error variable is anything other than 0, then the entire operation is skipped and the software goes back to its top-level loop. This is the same variable that becomes the error code passed along with the rest of the sensor data to the GUI. The error-reset button on the GUI is the only thing, short of a system power cycle, capable of resetting the error. Pushing that button sends the command to the SoC that sets the error variable back to zero.

The final significant upgrade to the software stemmed from a desire to speed up the transfer from one slide to another. The V1 Slide_Select function had the inner stack switch back and forth between moving rapidly and slowly. The stack would move 80 steps, then move only two steps at a time until the photogate sensor was triggered. The process would then repeat until the stack was at the desired slide's photogate trigger point. For example if starting at slide 1 and moving to slide 4 the stack would slowly move past two completely unnecessary photogate trigger points. This was initially added as a way to ensure that the system never overshoots the desired slide's photogate trigger point. For some slide changes, moving 80 steps was near perfect and not much time was added by this method. However, due to the geometry of the inner stack some trigger points were much more than 80 steps apart, meaning it would add a good deal of time to approach it at two-step intervals.

Since every possible current-to-desired slide combination had a different number

of steps between them it was proposed that the exact value be determined for each and placed into a matrix. This steps matrix could then be referenced to yield a single, quick motion of the inner stack. The steps matrix with all the exact values is represented in table 7 below.

Table 3.7

Steps Matrix						
Current Slide	5	0	112	214	314	416
	4	75	0	118	218	318
	3	180	80	0	110	212
	2	282	182	80	0	110
	1	386	286	184	82	0
		1	2	3	4	5
		Desired Slide				

By implementing the steps matrix the maximum switch time between slides was reduced from 2:00 minutes to 40 seconds.

3.5 Summary

The redesign of the software and creation of the user interfaces described above make the V2 a much more robust and versatile system than past iterations. By making all operations into functions and turning the main loop into a switch-case statement a user is no longer required to follow a specific path of operations. By incorporating the Manual control and GUI interfaces a user can also control the V2 with accuracy and ease. The establishment of two-way-communications between the GUI and SoC further this by displaying constant status and sensor data as well as error or warning messages.

Chapter 4

Performance Validation

4.1 Timing Verification

One of the top constraints for the NAFS was that it needed to be able to switch between any two filters in less than sixty seconds. This requirement is verified in Table 4.1 below. In very similar fashion to the steps matrix addressed in section 3.5, table 4.1 has a y-axis representing the currently inserted slide while the x-axis indicates the slide to be subsequently inserted.

Table 4.1

Switch Times (s)						
Current Slide	5	40.1	38.6	37.5	35.3	29.8
	4	38.7	34.9	34	28.5	35.3
	3	38	34.5	27.5	36	35.2
	2	36.4	30	34.9	36.5	40
	1	30.5	38	39.5	41.7	42.9
		1	2	3	4	5
		Desired Slide				

4.2 Position Verification

In order for photo-filters used within the NAFS to provide precise and consistent data, they must always have the same position and orientation within the telescope. If a filter is even slightly skewed from an earlier position, or the position of one of the other filters in the system, then natural particulates (dust) on the filters will not be able to be calibrated out of data with software.

Typically a flat field is taken with the filter in place as a sort of control image to identify the position of all dust spots. This image is then subtracted out from any actual data collected through the filter to mitigate the effects of said dust spots. However, if the NAFS were to place filters in a position not exactly the same as the one used during the flat field, then artifacts from the dust would remain in the final data.

It is because of this that the NAFS made position verification a top priority. To prove that the SoC interprets all position sensor data correctly and moves all filters to the same position, the location of every inserted filter was measured. This process was done twice to show consistency between all five filters and amongst each one individually.

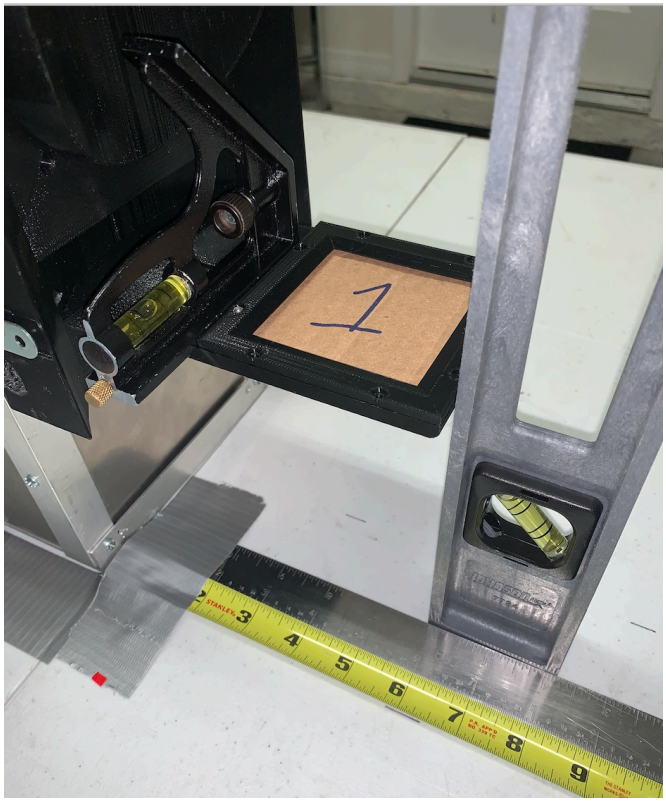

Figure 4.1: Slide 1 position test

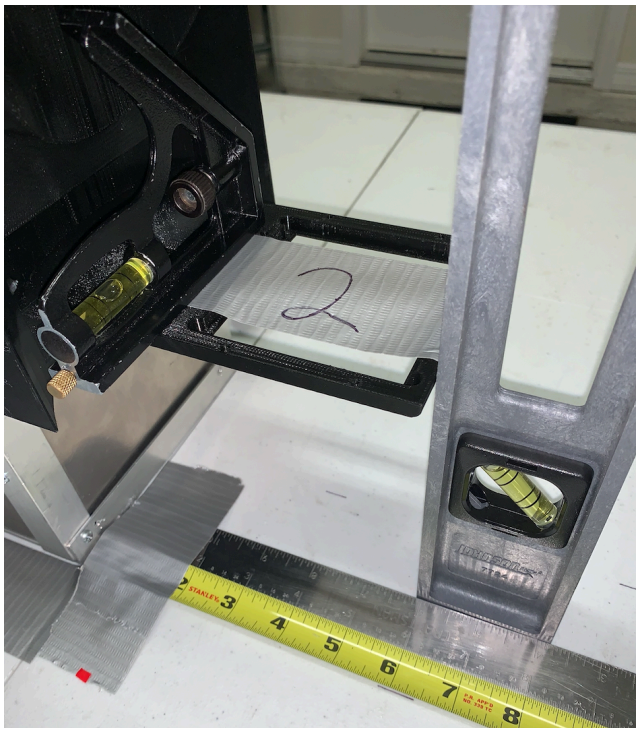

Figure 4.2: Slide 2 position test

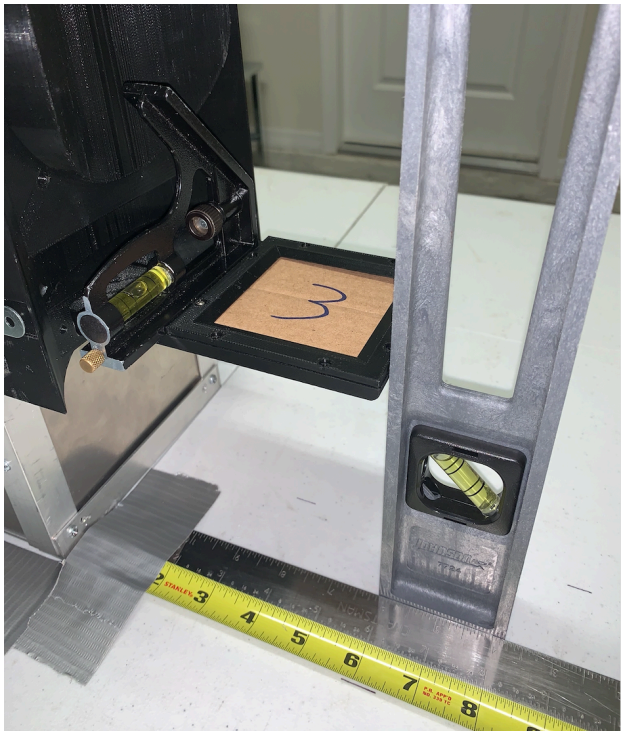

Figure 4.3: Slide 3 position test

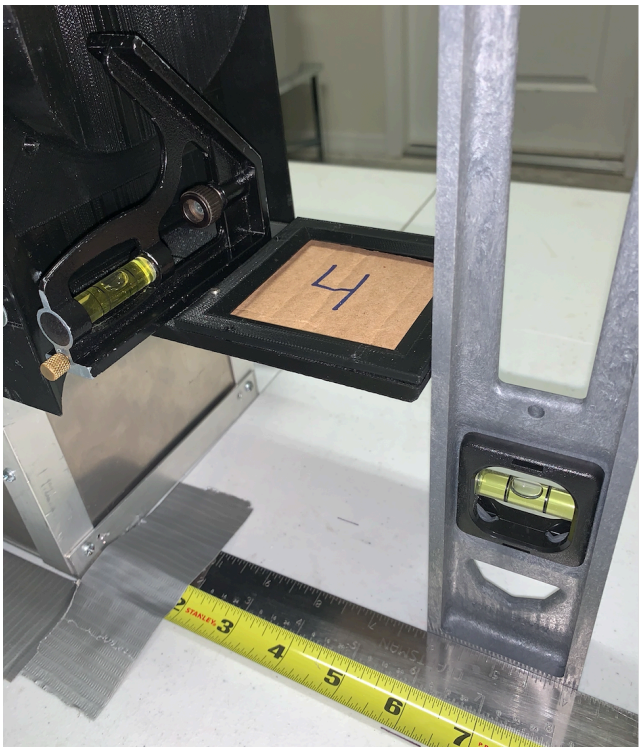

Figure 4.4: Slide 4 position test

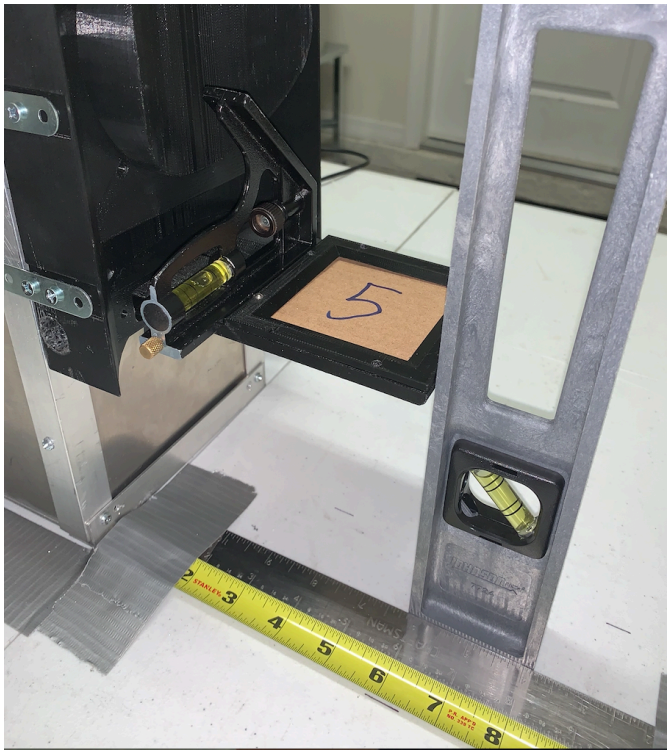

Figure 4.5: Slide 5 position test

Figures 4.1–4.5 displays the process that was undertaken to gather the results in Table 4.2. The system has a vertical and a horizontal degree of freedom (DoF), however, only the horizontal DoF is measured. This is due to the fact that if a slide were at an inaccurate vertical position, mechanical interferences would prevent it from moving horizontally at all. Thus by simply being extended out as shown in figures 4.1-4.5 the slide's vertical position is verified.

Table 4.2

	Trial		
Filter	1st	2nd	Δ
1	5.875	5.91	0.035
2	5.875	5.94	0.065
3	5.875	5.91	0.035
4	5.875	5.875	0
5	5.875	5.875	0
Std. Dev.	0.00000	0.02752	0.02752
	* All values are in Inches		

The results presented in table 4.2 show that the NAFS is definitively consistent in its placement of slides. While there is a standard deviation across all trials of 27.52 mils, the uncertainty of the measurement technique needs to be considered. As shown in figures 4.1-4.5 a ruler with 1/32" precision was used to make the measurements. Therefore all measurements have an uncertainty of 1/64" (16 mils). I estimate this to be a very conservative uncertainty due to trying to measure from an elevated position.

While the mechanics of the NAFS should allow for no rotational DoF, slight rotational motion was encountered at points during development. Steps were taken to mitigate this motion (see Appendix B, Inner Stack section) and this was verified using a level during the position testing. In figures 4.1-4.5 the level is visible on top of the slides and it is clear that all are leveled.

In order to further test the positioning performance of the NAFS I propose that a flat field test be performed. Instead of measuring the position of an inserted slide a flat field would be taken each time it is inserted. The difference of these flat fields could be taken to show any dust particulates out of place relative to the one another. Ideally nothing is out of place and a difference yields a perfect field with no features. Furthermore, this test should be conducted with the telescope system and NAFS at multiple orientations, to expose different force vectors to the device. Unfortunately, due to a lack of access to resources this test is not possible at this time.

4.3 Sensor Verification

It is through the joint use of an RFID sensor, Hall effect sensor, Photogate sensor, and stepper motors, that the position uniformity of all slides is capable. While the results in Table 4.2 imply that all sensors are functioning within acceptable parameters it is useful to show their read out values. These values are displayed in table 4.3 below and were collected during the first trial of the position verification test.

Table 4.3

Sensor Readings				
Filter	RFID	Photogate	Hall Effect	Step Diff.
1	1	27	14	25
2	2	27	14	15
3	3	27	14	15
4	4	27	14	15
5	5	28	18	13

The values read from the RFID sensor are the ID number of the tag it is reading. It is written into the SoC code to automatically convert the read ID number to a 1,2,3,4, or 5 depending on which ID it is.

It is important to note that the values returned from the photogate sensor are a direct function of ambient lighting conditions. A higher ambient will result in a higher value when triggered. In an untriggered state the sensor will return a value near 1000. During the time of which the values in table 4.3 were collected the photogate had an average untriggered value of 972. In order to account for the potential variance in ambient light the SoC will consider any value less than 100 to be a trigger.

Similar to the way the photogate values are interpreted, the Hall effect has an untriggered value of about 1400 and will trigger with any value less than 200. The variance of the triggered values is a function of magnet distance relative to the sensor face.

The step difference column describes the vertical distance in steps between the photogate tab that will cause a trigger and the actual slide insertion height. Once the photogate is triggered and the RFID tells which slide is aligned, the vertical stepper motor moves the inner stack down the respective amount of steps for that slide to properly align it with the insertion slot.

4.3 Summary

The tests discussed above were conducted as replacements for integrated telescope tests such as the flat-field differencing mentioned in section 4.2. Due to the COVID-19 pandemic and the ensuing laboratory closures the ability to integrate the NAFS V2 into Embry-Riddle's 1-meter telescope system was prevented. The timing, position and sensor tests were then designed to verify the V2 met all constraints while on a bench-top setup. A highest recorded switch time of 42.9s verifies the switch time constraint of 60s is met. While a standard deviation of 27.52 ± 15.63 mils in filter positioning shows that all filters are positioned precisely. The sensors test displaying nothing but expected values further shows that the V2 is operating nominally.

Chapter 5

System Summary

5.1 Final system depictions

Figures 5.1 through 5.4 show the final V2 system.

Figure 5.1: NAFS V2 final product; Exterior

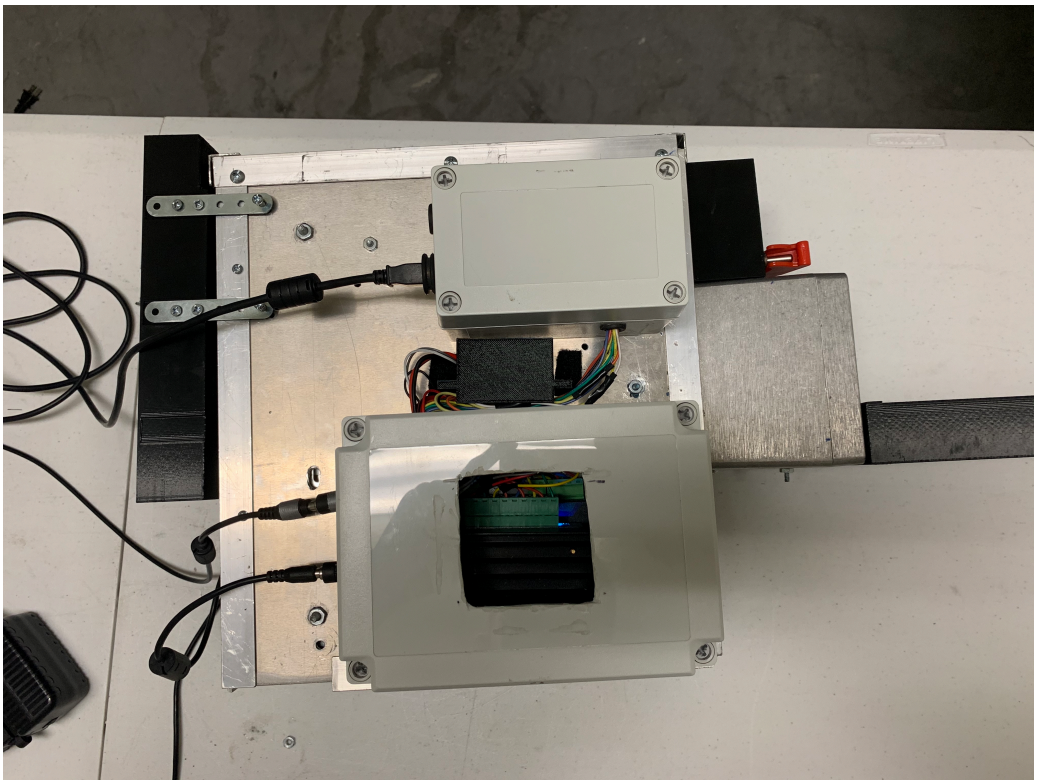

Figure 5.2: NAFS V2 final product; Wire management enclosures

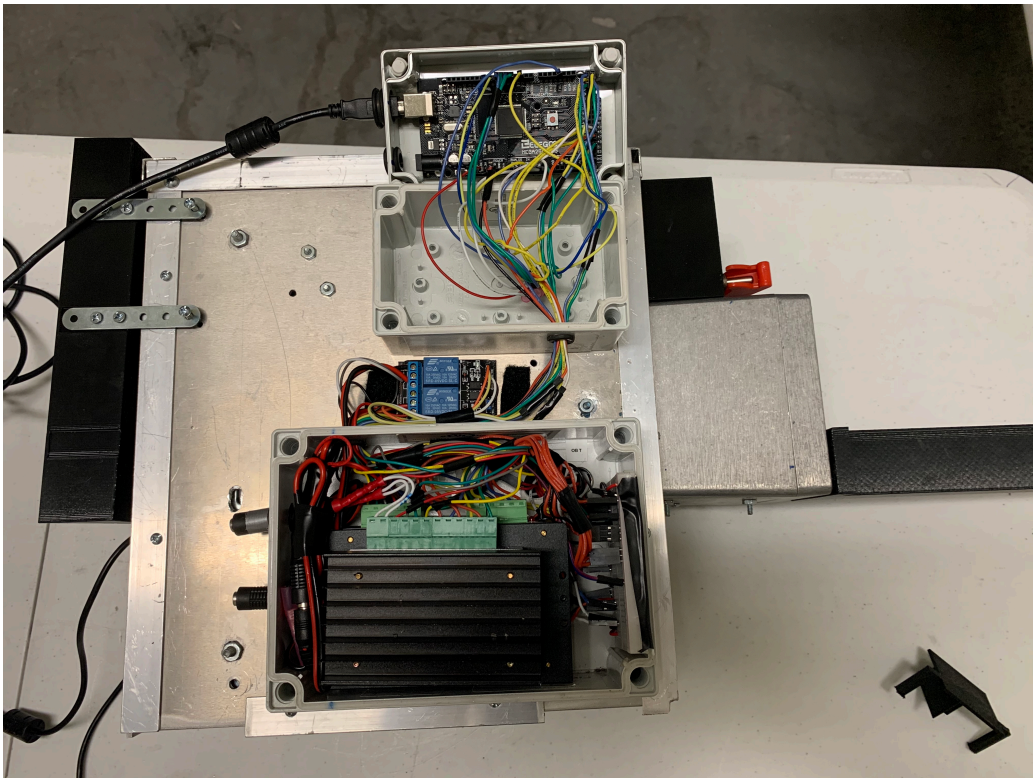

Figure 5.3: NAFS V2 final product; Wire management enclosures; cover off

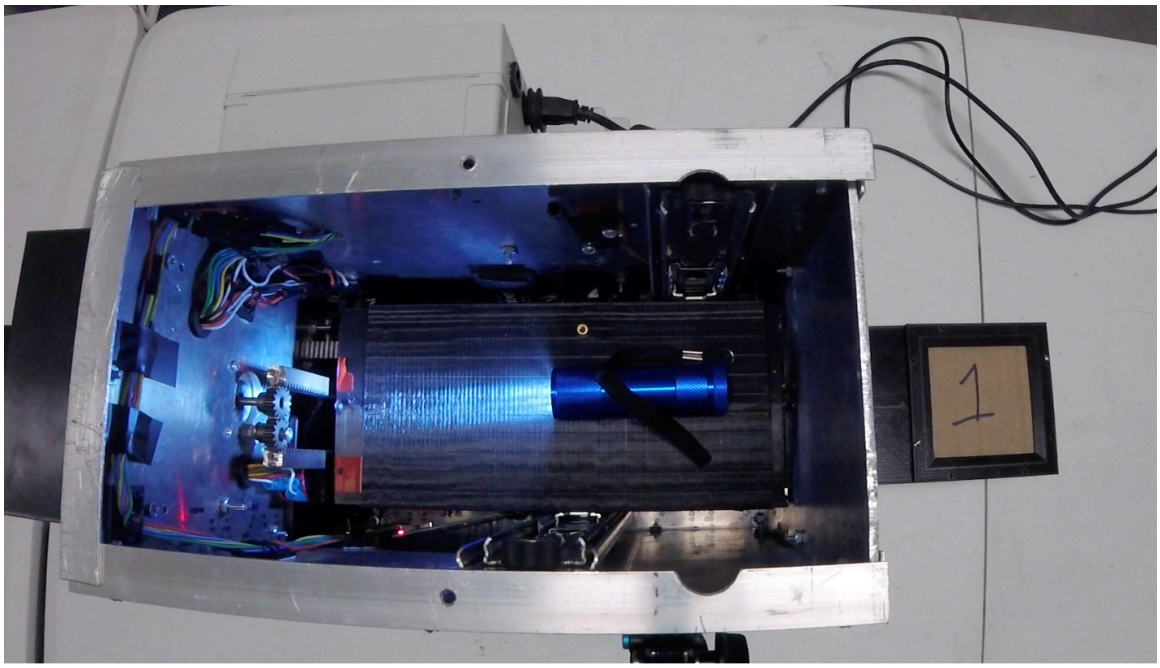

Figure 5.4: NAFS V2 final product; Top down view with filter 1 extended (top panel removed)

5.2 Filter changing scheme

The end result of V2 development was a system of motors and sensors that worked together seamlessly, to change filters. The steps that control the filter changing process start with a user input, either from the GUI, or from the manual control interface. Upon a user requesting a new slide the first thing the V2 does is check the Hall effect sensor to determine if a filter is currently inserted. If so the inserted slide is retracted before the process continues. After confirming that all slides have been successfully returned to within the inner stack vertical movement commences. Starting with two-step pulses the vertical motor drives the stack upward until the photogate is triggered. As soon as the photogate is triggered the RFID is checked to determine which slide the system currently has vertically aligned. Upon learning this the system will determine whether the inner stack needs to go up or down to get to the user requested slide and will reference the steps matrix to find the distance to this slide. Once the inner stack has traveled this distance it will again move upwards in two-step pulses until it hits a photogate and then check the RFID to confirm its at the right slide. The inner stack will then move downward a designated distance (see column 5 in table 4.3) to perfectly align the desired slide for insertion. After the pushrod extends completely and inserts the slide the Hall effect sensor is checked to verify correct horizontal placement.

If the GUI has been used to issue the command, after the button for the desired slide is selected the GUI begins to listen for any data from the SoC. As soon as it gets data it will display it on the screen for the user. The GUI will continue to update the

screen with incoming data until the requested operation is finished, at which point the interface becomes ready to accept another command.

The static friction brakes of both motors are toggled on/off at the beginning/end of any movement-based operation in order to allow motion when needed and keep the system locked in place at all other times. The relay, which is visible in figure 5.2 and 5.3 in between the two wire enclosures helps to facilitate the transfer of power between the static friction brakes and the motor drivers.

5.3 Conclusion

Overall the process described in section 5.2 was trying to be obtained from the start of BSEP senior design project. This final filter-changing scheme fits perfectly with the initial conceptual design that was sketched over a year and a half ago. Every modification presented in the chapters above were necessary contributors to the real world manifestation of that initial concept. Mechanical modifications made the system more robust and failures near obsolete. Software modifications have allowed the user to maximize the potential of the system by letting them perform the operations they want, when they need it. The verification tests have shown that while the system has improved in capability it has also further exceeded design requirements.

Appendix A

NAFS SETUP & OPERATION

The following is taken from the NAFS V2 User's Manual.

A.1 Installation

A.1.1 Parts

- Front interface mount
- Telescope apparatus mount screws (x6)
- Short metal screws (x5)
- Coupling Nuts ¼" x28 (x2)
- 2" long ¼" x28 bolt rod (x2)
- 0.625" ID, 2" OD washer (x2)
- 0.625" ID, 1" OD, ½" L Spacer (x2)
- 0.281" ID, 0.75 OD washer (x2)
- ¼" x28 hex nuts (x2)

A.1.2 Steps

The parts are placed onto the preexisting telescope apparatus mount's studs in the order listed above.

1. Remove the 2 persisting slot covers from the telescope apparatus mount
2. Slide the front interface mount on the telescope apparatus mount by lining up the circular extrusion on the interface with the circular slot on the mount
3. Secure the interface to the mount with the 6 mount screws
4. Screw a coupling nut halfway onto each of the two studs
5. Screw one 2" long bolt rods into each coupling nut until the rod and stud meet in the middle of the nut
6. Turn the NAFS onto its side and remove the bottom panel. To do this unscrew and remove the lower rear edge bracket and slide the panel out
7. Power the NAFS on and run the reload down function on the GUI (see Sections 2 and 3 for more details on this). Running this function will ask for confirmation that the lower panel has been removed

8. Once the inner stack has been disengaged from the motor, pull it all the way down
9. Rotate the black lever tab on the rail of the inner stack. By rotating the tab on each side at the same time and pulling the stack away from the outer housing the inner stack will be released from the rails and can be set aside
10. Lift the NAFS up to the telescope apparatus mount so that the bolt rods go through the holes in the top of the system and the front interface mount slot lines up with the slot on the front face of the NAFS
11. From the bottom, slide the larger washers over each of the bolt rods until they are flush with the inside of the top panel of the NAFS
12. Repeat the last step with the spacers, then the smaller washers, then the hex nuts
13. Tighten the hex nuts to secure the NAFS to the telescope apparatus mount
14. Screw the 5 brackets from the front interface to their holes on the outer housing
15. Slide the rails on the inner housing upwards into the outer housing's rails until they click
16. Select the "reload up" function of the GUI and lift the inner stack upwards until it meshes with the motor
17. Replace the NAFS's bottom panel and lower rear edge bracket

A.2 Hardware Connections

- Dual power cord into motor speed controllers (bottom right of the apparatus: viewed from the front)
- Arduino Power/Serial Connection (upper right of the apparatus: viewed from the front)
    - If using GUI: connect USB-C cable to PC
    - If not using GUI: Either USB-C or 9V cable will supply power

A.3 Setting up Communications

1. Determine which COM port on your PC the USB cable from the Arduino is plugged into
2. Select your COM port from the Serial Connection Setup (2) drop down box and hit "Connect"
3. If the connection is successful all the buttons on the Control tab will become active (not greyed out)

A.4 Standard Operation

Once a connection to the NAFS has been established all sensors and statuses will automatically update. The best way to tell the current state of the system is the RFID

sensor inside of the sensor suite (9) and the Horizontal Position label (12). These will tell you what if any slides are currently in the system.

At this point the GUI is ready for any user interaction. If a simple filter change is wanted, select the desired slide from the filter selection box (5). The system motion indicator light (7) will reflect that the device is now in motion and the button for the selected slide will change colors. If a slide is already in place the device will first remove it before moving to and inserting the desired one. Once the operation has been completed the indicator light will switch back to the blue stationary mode, but the button for the selected filter slide will remain a different color until the slide is removed or a new one is selected.

Six different unique operations are also available for selection within the Special Functions box (10). These are operations found useful during error mitigation or standard maintenance of the device. Descriptions of each operation are given in the table below.

Table A.1

Special Functions	
Function	**Description**
Home	Acts as reset for the system. Inner stack will move down slowly to find its bottom position, then moves back up to an alignment with slide 3. Resets the vertical position status bar along the way to ensure accuracy.
Brakes Off	Disables static friction brakes for both horizontal and vertical motors. Brakes can be turned back on by selecting any other motion based unction/operation or by cutting power to the system
Reload Up	Pulls the inner stack back up into standard position from its maintenance position. Vertical motor spins up in a pulsing motion until inner stack reaches slide 3 position. User will need to lift inner stack upward until the motor grabs it.
Reload Down	Pushes the inner stack down and out of the outer housing into its maintenance position. A warning box will appear prompting the user to remove the bottom panel of the outer housing before continuing. (See section 1 for details on how to remove bottom panel)
Extend Pusher	Moves the push rod horizontally to maximum extension
Retract Pusher	Moves the push rod horizontally to minimum extension

Should an error arise in the system it will be described both by an error message pop up and by the Errors box (6). The pop up message will give the error code, a description of the error, and some mitigation tips. The Errors box indicator light will change to red and the error code will be displayed next to it. Once the system encounters an error it will be unusable until the error reset button is selected. Only hit the error reset if you are sure the error has been mitigated.

As soon as the GUI is started an event log is initiated. Displayed on the third tab of the GUI the event log tracks all user initiated operations, system status changes, errors, and the time/date at which everything occurred. The log can be saved as a text file to the C:/TEMP directory by selecting the save button (1).

As the NAFS has no ON/OFF switch it is ok to cut power at any stationary state.

A.5 Manual Control

At the top of the outer housing's rear face sits the Manual Control Suite. Manual control is intended to be for quick, single operations, or for use in maintenance. If using the GUI concurrently with the manual control, check the manual control checkbox (3) on the communications tab of the GUI. This will allow all system statuses to continue to be updated even if the user command was initiated with the manual control.

To operate the manual control suite start by flipping the toggle switch with the red cover. This will give power to the LCD screen and the selection knobs. The left selection knob adjusts the resolution of the LCD screen to maximize performance in a variety of lighting conditions. The knob on the right is used to cycle through selectable operations. Once your desired operation is being displayed on the LCD screen, flip the toggle switch back by closing the red cover and the operation will be executed. The only new operation here is Resting Mode. This simply retracts any currently inserted slide and leaves the system completely inert until a new command is received.

Appendix B

MINOR MODIFICATIONS

Minor Modifications Tracking		
Subsystem	**Mod**	**Date (if recorded)**
Outer Housing	Made it easier to remove bottom panel by changing mounting screws for lower rear bracket	11/5/19
	Remove washers on rail mounting srews to reduce stress on inner stack	11/5/19
	Replaced electrical tape with permanent wire management brackets	11/18/19
	Create recesion for Hall effect to sit in to mitigate mechanical interference	12/3/19
	Put blue locktite on all mounting bolts	12/10/19
	Added cover for wire leash to prevent unplugging by inner stack movement	
	Reduce mounting toggle bolts from 5" to 3"	
	Added shim to left brackets on front interface mount to mitigate alignment issue	4/7/20
Inner Stack	Took 2 teeth off top of gear rack to mitigate interference with top of outer housing	11/21/19
	Epoxy slide linear glide rails in place to prevent minute rotational motion of slides	11/21/19
	File down epoxy that leaked during securing of linear guide rail. Excess was causing interferences	11/23/19
	Added multiple washers to inside of rail bolts to reduce interference of bolts and rails	
	Added bearings to slide 3's linear guide rail carriage to stabilize its motion	4/1/20
Motors	Added shim to inside of idle to give good fit on motor shaft	9/30/19
	Epoxy second neodymium magnet to pusher to increase lorentz force during slide retraction	11/21/19
	Added bracket to motor housing to support weight of pusher motor	

Appendix B: Minor Modifications

	Removed more excess epoxy preventing full sldie insertion	
	Tightened down pusher gear & secured it to motor shaft with epoxy	
	Implemented a much better set screw on pushed gear to prevent slipping	4/7/20
Sensors	Make low profile wiring for Hall effect sensor	11/21/19
	Switch from tape to epoxy for Hall effect sensor securement	11/21/19
	Flip Hall effect/magnets over to fix polarity issue	12/2/19
	Move Hall effect 0.5" left to get closer to magnets	12/2/19
	Added brace to photogate to prevent from rotating out of position	
	Place Hall effect magnets at the exact same position on every slide	
	Added wire leash to bottom micro switch due to reload functions	
Software: SoC	Added serial reads from GUI	11/11/19
	Implimented MCU into main loop	11/11/19
	Packaged entire sending data process into one function	11/12/19
	Added system status parameters to the message for the GUI	11/12/19
	Redefined function parameters and returns for all functions to optimize code	11/12/19
	Added '#' character as designate for incoming command	11/15/19
	Added ability to turn off horizontal and vertical brakes seperatly	
	Added some special operations to MCU	
	Added short sequence prior to every vertical movement to identify exactly which filter device is at. Done because RFID 3 was weaker than the others and can only be read at one spot	
	Added 2nd attempt slide retrival sequence	
	Made reload sequences no longer count towards vertical position tracking	
	Took mid-movement sensor data transmission out as it was adding unneccesary time	
	Optimized slide_select function to reduce filter switch time	4/7/20
Software: GUI	Added message box warning on reload up/down click	11/10/19
	Added update sensor button	11/11/19

Appendix B: Minor Modifications

	Added up/down in/out indicator light	11/21/19
	Added x/y steps tracker to make easier to understand system position	11/21/19
	Switch colors of moving indicator light so that red=moving	
	Put the error description in error message box	
	Added in "Manual Control Checkbox" so that system status still gets updated while in manual operation	
	Added log file	
	Time/date stamped event log entries	
	Created save button for Event log. Saves to "TEMP" dicrectory	
	Time/date stamped event log file name when saved	
	Changed horizontal position progress bar to a simple status display	
	Made it so an activated slide turns a different color	

Appendix C

SoC SOFTWARE CODE

```
/*  1m Filter Aparatus Control
 *  Embry-Riddle Aeronautical University
 *  Author: Tyler Neusch
 *  Credit to : Adam Campagnolo, Dejan Nedelkovski
 */

#include <LiquidCrystal.h> //includes the LiquidCrystal Library (LCD
Display)
#include <RFID.h> //includes RFID Library for Mega
#include <SPI.h>  //includes SPI Library

//Setup RFID
//Define unsigned variable types
#define  uchar unsigned char
#define uint  unsigned int

uchar fifobytes;
uchar fifoValue;
#define SDA_DIO 9
#define RESET_DIO 8

/* Create an instance of the RFID library */
RFID RC522(SDA_DIO, RESET_DIO);

//Maximum length of the array
#define MAX_LEN 16

//End Setup RFID

LiquidCrystal lcd(18, 2, 4, 5, 6, 7); // Creates an LC object.
Parameters: (rs, enable, d4, d5, d6, d7);

//Define Mega Pin references
const int gate_Stack = A6;  //Define Photogate analog input
const int Selection_knob = A5;  //define knob anolog input
const int Button_pin = A3;  //define button input digital
const int Display_pin = A4; //define backlight control pin
const int OB_Top_pin = 44;  //define Out of Bounds Top pin
const int OB_Bot_pin = 46;  //define Out of Bounds Bottom pin
const int Hall_effect = A7; //define hall effect sensor pin
const int Relay_Pin = 45; //define relay control pin
const int Brake_pin = 47; //define relay control pin for friction brake

const int PUL_big = 22; //Define pulse pin for stack motor
const int DIR_big = 24; //Define direction pin for stack motor
const int ENA_big = 26; //Define enable pin for stack motor
```

```
const int PUL_small = 25; //Define pulse pin for pusher rod motor
const int DIR_small = 23; //Define direction pin for pusher rod motor
const int ENA_small = 27; //Define enable pin for pusher rod motor

//Define variables to be used in algorithm
int
start_up=0,set_up=0,desired_slide=0,current_slide=0,vert_offset=0,verti
cal_distance,rfid_Test,rfid_value,homed,task,error_code=0;
int btwn_slide_steps=2550,retract_command=0,Manual_Selection
=0,push_distance=800,trash=0;
int x_pos = 0, y_pos =0;
int manual_control=0,hall_eff=0;
bool
lined_up=false,dir_var=HIGH,OB_Top=0,OB_Bot=0,out_of_bounds_top=0,out_o
f_bounds_bot=0;
bool
error=false,pusher_extended=false,home_position=false,cs=LOW,ds=LOW;
volatile bool Button = LOW;
char gui_request;
```

```
//Setup===============================================================
====================================
void setup() {
  // put your setup code here, to run once:
lcd.begin(16,2); // Initializes the interface to the LCD screen, and
specifies the dimensions (width and height) of the display

//Declare Pinmodes
  pinMode (gate_Stack, INPUT);
  pinMode (Selection_knob, INPUT);
  pinMode (Button_pin, INPUT);
  pinMode (Display_pin, OUTPUT);
  pinMode (OB_Top_pin, INPUT);
  pinMode (OB_Bot_pin, INPUT);
  pinMode (Hall_effect, INPUT);
  pinMode(Relay_Pin, OUTPUT);
  pinMode (Brake_pin, OUTPUT);

  pinMode (PUL_big, OUTPUT);
  pinMode (DIR_big, OUTPUT);
  pinMode (ENA_big, OUTPUT);

  pinMode (PUL_small, OUTPUT);
  pinMode (DIR_small, OUTPUT);
  pinMode (ENA_small, OUTPUT);


//Configure Serial Port(s)
  Serial.begin(9600);
  Serial.setTimeout(1000);
```

```
//Configure SPI Bus
  SPI.begin();

//Initialize the RFID reader
  RC522.init();

//Enable button interupt function
  attachInterrupt(0,pin_ISR,CHANGE);

  digitalWrite(Display_pin, LOW); // turn backlight off. Replace 'HIGH'
with 'LOW' to turn it off.

//Turn Brakes On
         digitalWrite(Brake_pin, LOW);  //Engage Brakes
        //Serial.println("Locked");
        delay(1000);
        digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power

//Print System Startup
Serial.println("SoC Startup Complete");

} //End Setup

//Main
Loop================================================================
======================
void loop()
{
  // put your main code here, to run repeatedly:
  //Check to see if manual control has been activated

  Button = digitalRead(Button_pin);
  if (Button == 1)
  {
    digitalWrite(Display_pin, HIGH);
    while (Button == 1)
    {
       task = ManualSelection();
       Button = digitalRead(Button_pin);
       //Serial.println("MS Mode Open");
    }
    //Serial.println("MS Mode Close");
    //Serial.println(desired_slide);
    if (task <=5)
    {
      task = desired_slide;
      slide_select(desired_slide);
      if (Serial.available() > 0)
      {
        sensors_f();
      }
    }
    else
    {
      man_sel_task(task);
    }
    digitalWrite(Display_pin, LOW);
```

```
 }

//Retrieve command request from gui
if (Serial.available() > 0)
{
gui_request = UserInput();

switch (gui_request)
{
  case 'N':
    //Update Sensors
    //Serial.println("Sierra");
    sensors_f();
    break;
  case 'E':
    //Home
    homed = initialize();
    sensors_f();
    Serial.println(homed);
    break;
  case 'U':
    //Reload Down
    reload(0);
    sensors_f();
    Serial.println("Reload down");
    break;
  case 'S':
    //Reload Up
    reload(1);
    sensors_f();
   Serial.println("Reload up");
    break;
  case 'C':
    //Extend Pusher
    digitalWrite(Relay_Pin, HIGH); //Apply power to motors
    digitalWrite(ENA_big, HIGH);
    delay(1000);
    digitalWrite(Brake_pin, HIGH); //disengage brakes

    MovePusher(HIGH,13700);  //DETERMINE STEPS NEEDED

    digitalWrite(Brake_pin, LOW);  //Engage Brakes
    delay(1000);
    digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
    sensors_f();
    break;
  case 'H':
    //Retract Pusher
    digitalWrite(Relay_Pin, HIGH); //Apply power to motors
    digitalWrite(ENA_big, HIGH);
    delay(1000);
    digitalWrite(Brake_pin, HIGH); //disengage brakes
    MovePusher(LOW,13700);  //DETERMINE STEPS NEEDED
       digitalWrite(Brake_pin, LOW);  //Engage Brakes
      //Serial.println("Locked");
      delay(1000);
      digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
```

```
      sensors_f();
      break;
    case 'T':
      //All Brakes off
      digitalWrite(Brake_pin, HIGH); //disengage brakes
      digitalWrite(Relay_Pin, LOW); //Disable power to motors
      digitalWrite(ENA_big, LOW);
      break;
    case 'a':
      //Slide 1 select
      slide_select(1);
      sensors_f();
      //Serial.println("Slide 1");
      break;
    case 'b':
      //Slide 2 select
      slide_select(2);
      sensors_f();
      //Serial.println("Slide 2");
      break;
    case 'd':
      //Slide 3 select
      slide_select(3);
      sensors_f();
      break;
    case 'f':
      //Slide 4 select
      slide_select(4);
      sensors_f();
      break;
    case 'g':
      //Slide 5 select
      slide_select(5);
      sensors_f();
      break;
    case 'X':
    //Brakes on
         digitalWrite(Brake_pin, LOW);  //Engage Brakes
        //Serial.println("Locked");
        delay(1000);
        digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
    break;
    case 'Z':
    //clear error
      error_code = 0;
      sensors_f();
    break;
  }
  }

}


//User Defined
Functions=============================================================
===============
```

```
//
  void pin_ISR() {
    Button = digitalRead(Button_pin);
    digitalWrite(Display_pin,Button);
  }
//------------------------------------------
 char UserInput(){
  //Serial.println("Inside User Input func");
  char b;
  int cont = 1;
  //while(Serial.available() <= 0) {}
  while (cont == 1)
  {
    delay(500);
    //Serial.println("waiting for #");
    if (Serial.peek() == '#')
    {
       Serial.read();
       b = Serial.read();
       cont = 0;
       //Serial.println(b);
       delay(500);
    }
    else
    {
      Serial.println("# not found");
      Serial.read();
    }
  }
  //Serial.println("Outside of while loop");
  Serial.end();
  Serial.begin(9600);
  return b;
  }
  //------------------------------------------
  int PhotoGate_Stack(){

  int g=0;
  //Check Gate
  g = analogRead(gate_Stack);
  return g;
}
//------------------------------------------
void MoveStack(bool dir_var, int vertical_distance){

   digitalWrite(Relay_Pin, HIGH); //Apply power to motors
   digitalWrite(ENA_big, HIGH);
   delay(1000);
   digitalWrite(Brake_pin, HIGH); //disengage brakes

    //Run Large Motor to drive stack some vertical distance
      for (int c = 0; c < vertical_distance; c++)
      {
        OB_Top = digitalRead(OB_Top_pin);
        OB_Bot = digitalRead(OB_Bot_pin);

        digitalWrite(DIR_big, dir_var);
```

```
        digitalWrite(ENA_big, HIGH);
        digitalWrite(PUL_big, HIGH);
        delayMicroseconds(3000);
        digitalWrite(PUL_big, LOW);
        delayMicroseconds(9000);

        if(OB_Top != 0)
        {
        out_of_bounds_top = 1;
        }
        else if(OB_Bot != 0)
        {
          //out_of_bounds_bot = 1;
        }
        if (dir_var == HIGH)
        {
          y_pos = y_pos +1;
        }
        else if (dir_var == LOW)
        {
          y_pos = y_pos -1;
        }
        if (c % 25 == 0)
        {
         //sensors();
        }
      }
  }
//  ------------------------------------------------------
void MoveStack_1(bool dir_var, int vertical_distance){

   digitalWrite(Relay_Pin, HIGH); //Apply power to motors
   digitalWrite(ENA_big, HIGH);
   delay(1000);
   digitalWrite(Brake_pin, HIGH); //disengage brakes

    //Run Large Motor to drive stack some vertical distance
      for (int c = 0; c < vertical_distance; c++)
      {
        OB_Top = digitalRead(OB_Top_pin);
        OB_Bot = digitalRead(OB_Bot_pin);

        digitalWrite(DIR_big, dir_var);
        digitalWrite(ENA_big, HIGH);
        digitalWrite(PUL_big, HIGH);
        delayMicroseconds(1000);
        digitalWrite(PUL_big, LOW);
        delayMicroseconds(1000);

        if(OB_Top != 0)
        {
        out_of_bounds_top = 1;
        }
        else if(OB_Bot != 0)
        {
          //out_of_bounds_bot = 1;
        }
```

```
      }
  }
//  ------------------------------------------------
int RFID() {
  /* Has a card been detected? */
  if (RC522.isCard())
  {
    /* If so then get its serial number */
    RC522.readCardSerial();
                // Should really check all pairs, but for now we'll
just use the first
            if(RC522.serNum[0] == 38)                       //You can
change this to the first byte of your tag by finding the card's ID
through the Serial Monitor
            {
                //Serial.print("\nSlide One");
                current_slide = 1;
            } else if(RC522.serNum[0] == 86) {              //You can
change this to the first byte of your tag by finding the card's ID
through the Serial Monitor
                //Serial.print("\nSlide Two");
                current_slide = 2;
            } else if(RC522.serNum[0] == 150) {              //You can
change this to the first byte of your tag by finding the card's ID
through the Serial Monitor
                //Serial.print("\nSlide Three");
                current_slide = 3;
            } else if(RC522.serNum[0] == 214) {              //You can
change this to the first byte of your tag by finding the card's ID
through the Serial Monitor
                //Serial.print("\nSlide Four");
                current_slide = 4;
            } else if(RC522.serNum[0] == 230) {              //You can
change this to the first byte of your tag by finding the card's ID
through the Serial Monitor
                //Serial.print("\nSlide Five");
                current_slide = 5;
            }

      //lcd.print("Slide ");
      //lcd.print(current_slide);
      //lcd.setCursor(2,1);
      //lcd.print("Detected");
      //Serial.print("Slide ");
      //Serial.print(current_slide);
      //Serial.println(" Detected!");
      //digitalWrite(Display_pin,HIGH);
  }
  delay(250);

  return current_slide;
  }
  //-------------------------------------------------------------------
--------------
      void MovePusher(bool dir_var, int vertical_distance){

    //Run Small Motor to push slide some horizontal distance
```

```
        digitalWrite(DIR_small, dir_var);
        digitalWrite(ENA_small, HIGH);
      for (int c = 0; c < vertical_distance; c++)
      {
        digitalWrite(PUL_small, HIGH);
        delayMicroseconds(250);
        digitalWrite(PUL_small, LOW);
        delayMicroseconds(250);

        if (dir_var == HIGH)
        {
          x_pos = x_pos +1;
        }
        else if (dir_var == LOW)
        {
          x_pos = x_pos -1;
        }
        if (c % 50 == 0)
        {
          //sensors();
        }
        //}
      }
  }
  //-------------------------------------------------------------------------
----------
  int ManualSelection (){

    Manual_Selection = analogRead(Selection_knob);  //Read
potentiometer to determine indicated slide

    //Display selection on LCD Display
     if(Manual_Selection >= 893 && Manual_Selection <= 1023){
      lcd.print("Slide One");
      lcd.setCursor(2,1);
      //lcd.print("H-Alpha");
      vertical_distance = 1000;
      desired_slide = 1;
     }
     else if(Manual_Selection >=766 && Manual_Selection <893){
      lcd.print("Slide Two");
      lcd.setCursor(2,1);
      //lcd.print("Red");
      vertical_distance = 1000;
      desired_slide = 2;
     }
      else if(Manual_Selection >=639 && Manual_Selection <766){
      lcd.print("Slide Three");
      lcd.setCursor(2,1);
      //lcd.print("Blue");
      vertical_distance = 1000;
      desired_slide = 3;
     }
      else if(Manual_Selection >=512 && Manual_Selection <639){
      lcd.print("Slide Four");
      lcd.setCursor(2,1);
```

```
      //lcd.print("Green");
      vertical_distance = 1000;
      desired_slide = 4;
     }
      else if(Manual_Selection >=385 && Manual_Selection <512){
      lcd.print("Slide Five");
      lcd.setCursor(2,1);
      //lcd.print("Polarized");
      vertical_distance = 6000;
      desired_slide = 5;
     }
      else if(Manual_Selection >=289 && Manual_Selection <385){
      lcd.print("Pusher Brakes");
      lcd.setCursor(2,1);
      lcd.print("OFF");
      desired_slide = 6;

     }
      else if(Manual_Selection >=193 && Manual_Selection <289){
      lcd.print("Vertical Brakes");
      lcd.setCursor(2,1);
      lcd.print("OFF");
      desired_slide = 7;

     }
     else if(Manual_Selection >=97 && Manual_Selection <193){
      lcd.print("All Brakes");
      lcd.setCursor(2,1);
      lcd.print("ON");
      desired_slide = 8;

     }
      else if(Manual_Selection >=0 && Manual_Selection <97){
      lcd.print("Resting Mode");
      desired_slide = 9;
     }

   delay(25);

   lcd.clear();

   return desired_slide;
    }
//-------------------------------------------------------------------------
-----------------
int hall_effect(){

  int value = 0;

  value = analogRead(Hall_effect);

  return value;
}
//-------------------------------------------------------------------------
-----------------
int initialize(){
  int homed;
```

```
int phtgt;

OB_Bot = digitalRead(OB_Bot_pin);

digitalWrite(Relay_Pin, HIGH); //Enable Motor Driver Power
delay(1000);
digitalWrite(Brake_pin, HIGH); //Disable Breaks
 while( OB_Bot != 1)
{
  //Serial.println("Homing");
  MoveStack(LOW,2);
  sensors();
}

if(OB_Bot == 1)
{
  y_pos = 0;
  sensors();
  MoveStack(HIGH,187);
}

do
{
  current_slide = RFID();
  phtgt = PhotoGate_Stack();
  //Serial.println(current_slide);
  //Serial.println(phtgt);
  MoveStack(HIGH,2);
  sensors();
}while(current_slide != 3 && PhotoGate_Stack() >= 100);

  while( PhotoGate_Stack() >= 100)
   {
      MoveStack(HIGH,1);
      sensors();
   }
 if (PhotoGate_Stack() < 100 && RFID()==3)
 {
  error_code = 0;
 }
 else
 {
  error_code = 4;
 }
sensors();
MoveStack(LOW,15);
sensors();
current_slide = RFID();
//Serial.println(current_slide);
homed =1;

if( current_slide != 1 && current_slide != 5)
   {
      digitalWrite(Brake_pin, LOW);  //Engage Brakes
      //Serial.println("Locked");
      delay(1000);
      digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
```

```
      }

  //Initialization Complete
  //Serial.println("Initialization Complete!");
  x_pos = 0;
return homed;
}

//--------------------------------------------------------------------------
---------------------
void slide_select(int desired_slide)
{

//Initialize Variables
  int steps_matrix[5][5] = {{0,112,214,314,416},
                            {75,0,118,218,318},
                            {180,80,0,110,212},
                            {282,182,80,0,110},
                            {386,286,184,82,0}};
  int cy =0;
  int dx =0;
  int error =0;
  int adj =0;
  int current_slide_func;

  //Retract pusher if already inserted
  hall_eff = analogRead(Hall_effect);
 if (hall_effect() < 100)
      {
              //Retract Pusher
      digitalWrite(Relay_Pin, HIGH); //Apply power to motors
      digitalWrite(ENA_big, HIGH);
      delay(1000);
      digitalWrite(Brake_pin, HIGH); //disengage brakes

      MovePusher(LOW,13700);  //DETERMINE STEPS NEEDED
      }
    //2nd retrival attempt if needed
    if (hall_effect() < 100)
      {
                //Extend Pusher
      digitalWrite(Relay_Pin, HIGH); //Apply power to motors
      digitalWrite(ENA_big, HIGH);
      delay(1000);
      digitalWrite(Brake_pin, HIGH); //disengage brakes

      MovePusher(HIGH,13700);  //DETERMINE STEPS NEEDED
      MovePusher(LOW,13700);
      }
        digitalWrite(Brake_pin, LOW);  //Engage Brakes
        //Serial.println("Locked");
        delay(1000);
        digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
    if (hall_effect() < 100)
    {
      error_code = 1;
    }
```

```
  //continue with new filter selection
  if (error_code == 0)
  {

  //Verify position with RFID
  while (PhotoGate_Stack() >= 100)
  {
   MoveStack_1(HIGH,2);
  }
  Serial.print("At Current photogate....");
  sensors();
  current_slide_func = RFID();

  //Adjust vert alignment
  if( current_slide_func == 4)
  {
    adj =15;
  }
  if( current_slide_func == 1)
  {
    adj =25;
  }
  if( current_slide_func == 5)
  {
    adj =15;
  }
  if( current_slide_func == 3)
  {
    adj =15;
  }
  if( current_slide_func == 2)
  {
    adj =15;
  }

//Vertical Movement using steps matrix
//Set direction variable
if(current_slide_func >= desired_slide)
    {
      //Serial.println("C > D");
      dir_var = LOW;
    }
    else if(current_slide_func < desired_slide)
    {
      //Serial.println("D > C");
      dir_var = HIGH;
    }
  vert_offset = abs(current_slide_func - desired_slide);
//Move vertically (Slew)
   digitalWrite(Relay_Pin, HIGH); //Apply power to motors
   delay(1000);
   digitalWrite(Brake_pin, HIGH); //disengage brakes
   cy = current_slide_func - 1;
   dx = desired_slide - 1;
   if (dir_var == HIGH)
   {
```

```
    MoveStack(dir_var,steps_matrix[cy][dx] - adj);
   }
   else if (dir_var == LOW)
   {
    if (current_slide_func == desired_slide)
    {
      MoveStack(dir_var,adj);
    }
    else
    {
    MoveStack(dir_var,steps_matrix[cy][dx] + adj);
    }
   }
   sensors();
//Move vertically (Fine)
if (current_slide_func != desired_slide)
{
    while (PhotoGate_Stack() >= 100)
  {
   MoveStack_1(dir_var,2);
  }
  Serial.print("At Desired Photogate....");
  sensors();

  if (RFID() != desired_slide)
  {
    error_code = 3;
  }
  else
  {
      if( desired_slide == 4)
      {
         MoveStack(LOW,15);
      }
      if( desired_slide == 1)
      {
         MoveStack(LOW,25);
      }
      if( desired_slide == 5)
      {
         MoveStack(LOW,15);
      }
      if( desired_slide == 3)
      {
         MoveStack(LOW,15);
      }
      if(desired_slide == 2)
      {
        MoveStack(LOW,15);
      }
  }
}
  lined_up = true;
  current_slide_func = RFID();

  //Correct vertical position achieved and verified
  if (lined_up == true && error_code == 0)
```

```
  {
    MovePusher(HIGH,13700);  //DETERMINE STEPS NEEDED

    digitalWrite(Brake_pin, LOW);  //Engage Brakes
    delay(1000);
    digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power

    hall_eff = analogRead(Hall_effect);
    //Serial.println(hall_eff);

    if( hall_eff < 800)
    {
    pusher_extended = true;
    }
    else
    {
      //error = true;
      error_code =2;
      //Serial.println("ERROR, slide not inserted!");
    }
  }
  }
}
//-------------------------------------------------------------------------
------------------------
void reload(int UD)
{
    int pos;

    digitalWrite(Relay_Pin, HIGH); //Apply power to motors
    delay(1000);
    digitalWrite(Brake_pin, HIGH); //disengage brakes

    if (UD == 0)
    {
      //Move Stack Down
      MoveStack(LOW,1000);  //DETERMINE STEPS NEEDED
    }
    if (UD == 1)
    {
      //Move Stack Up
      while (pos !=5)
      {
         MoveStack(HIGH,50);  //DETERMINE STEPS NEEDED
         pos = RFID();
         delay(500);
      }
    }
    digitalWrite(Brake_pin, LOW);  //Engage Brakes
    delay(1000);
    digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
    y_pos = 0;
}
//-------------------------------------------------------------------------
----------------------------
void sensors()
{
```

```
  int rfid;
  int pg;
  int hall;
  int tms;
  int bms;
  double vps;
  double hps;
  int vps_round;
  int hps_round;
  char mcm;

  rfid = RFID();
  pg = PhotoGate_Stack();
  hall = hall_effect();
  tms = digitalRead(OB_Top_pin);
  bms = digitalRead(OB_Bot_pin);
  vps = y_pos / 3.94;
  hps = x_pos / 142;
  vps_round = round(vps);
  hps_round = round(hps);
  if (vps_round <0)
  { vps_round = 0;}
    if (hps_round <0)
  { hps_round = 0;}

  Serial.print("*");
  Serial.print(rfid);
  Serial.print("^");
  Serial.print(pg);
  Serial.print("^");
  Serial.print(hall);
  Serial.print("^");
  Serial.print(tms);
  Serial.print("^");
  Serial.print(bms);
  Serial.print("^");
  Serial.print(vps_round);
  Serial.print("^");
  Serial.print("50");
  Serial.print("^");
  Serial.print(error_code);
  Serial.println("!");
}

//--------------------------------------------------------------------------
-----------------------------
void sensors_f()
{
  int rfid;
  int pg;
  int hall;
  int tms;
  int bms;
  double vps;
  double hps;
  int vps_round;
  int hps_round;
```

```
  char mcm;

  rfid = RFID();
  pg = PhotoGate_Stack();
  hall = hall_effect();
  tms = digitalRead(OB_Top_pin);
  bms = digitalRead(OB_Bot_pin);
  vps = y_pos / 3.94;
  hps = x_pos / 142;
  vps_round = round(vps);
  hps_round = round(hps);
    if (vps_round <0)
  { vps_round = 0;}
    if (hps_round <0)
  { hps_round = 0;}

  Serial.print("*");
  Serial.print(rfid);
  Serial.print("^");
  Serial.print(pg);
  Serial.print("^");
  Serial.print(hall);
  Serial.print("^");
  Serial.print(tms);
  Serial.print("^");
  Serial.print(bms);
  Serial.print("^");
  Serial.print(vps_round);
  Serial.print("^");
  Serial.print("50");
  Serial.print("^");
  Serial.print(error_code);
  Serial.print("!");
  Serial.println("$");
}
//--------------------------------------------------------------------------
---------------------
void man_sel_task(int task)
{
  if (task == 6)
  {
    //Pusher brakes off
      digitalWrite(Relay_Pin, HIGH); //Enable power to motors
      digitalWrite(ENA_big, HIGH);
      delay(1000);
      digitalWrite(Brake_pin, HIGH); //disengage brakes
  }
  else if (task == 7)
  {
    //All brakes off
      digitalWrite(Brake_pin, HIGH); //disengage brakes
      digitalWrite(Relay_Pin, LOW); //Disable power to motors
      digitalWrite(ENA_big, LOW);
  }
  else if (task == 8)
  {
    //All brakes on
```

```
      digitalWrite(Brake_pin, LOW);  //Engage Brakes
      //Serial.println("Locked");
      delay(1000);
      digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
  }
  else if (task == 9)
  {
    //Resting Mode
    if (hall_effect() < 100)
      {
              //Retract Pusher
      digitalWrite(Relay_Pin, HIGH); //Apply power to motors
      digitalWrite(ENA_big, HIGH);
      delay(1000);
      digitalWrite(Brake_pin, HIGH); //disengage brakes

      MovePusher(LOW,13700);  //DETERMINE STEPS NEEDED
      }
      while (hall_effect() < 100)
      {
                //Extend Pusher
      digitalWrite(Relay_Pin, HIGH); //Apply power to motors
      digitalWrite(ENA_big, HIGH);
      delay(1000);
      digitalWrite(Brake_pin, HIGH); //disengage brakes

      MovePusher(HIGH,13700);  //DETERMINE STEPS NEEDED
      MovePusher(LOW,13700);
      }
        digitalWrite(Brake_pin, LOW);  //Engage Brakes
        //Serial.println("Locked");
        delay(1000);
        digitalWrite(Relay_Pin, LOW);   //Disable Motor Driver Power
  }
}
```

Appendix D

GUI SOFTWARE CODE

```
using System;
using System.IO;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.IO.Ports;

namespace Thesis1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
            s1_button.Enabled = false;
            s2_button.Enabled = false;
            s3_button.Enabled = false;
            s4_button.Enabled = false;
            s5_button.Enabled = false;
            home_button.Enabled = false;
            brakes_button.Enabled = false;
            update_sensors_button.Enabled = false;
            rp_button.Enabled = false;
            ep_button.Enabled = false;
            rd_button.Enabled = false;
            ru_button.Enabled = false;
            MCM_CB.Enabled = false;
            error_rst.Enabled = false;
            indicator.Text = ("Stationary");
            indicator.BackColor = Color.Blue;
            pusher_indicator.BackColor = Color.White;
            pusher_indicator.Text = "Retracted";
            serial_indicator.Text = "Not Connected";
            serial_indicator.BackColor = Color.Red;
        }


        private void label1_Click(object sender, EventArgs e)
        {

        }
```

```
        private void statusStrip1_ItemClicked(object sender,
ToolStripItemClickedEventArgs e)
        {

        }

        private void label7_Click(object sender, EventArgs e)
        {

        }

        private void label15_Click(object sender, EventArgs e)
        {

        }

        private void comboBox1_SelectedIndexChanged(object sender, EventArgs
e)
        {
            string[] ports = SerialPort.GetPortNames();

        }

        private void CDC_Click(object sender, EventArgs e)
        {
            if (serialPort1.IsOpen)
            {
                serialPort1.Close();
                CDC.Text = ("Connect");
                s1_button.Enabled = false;
                s2_button.Enabled = false;
                s3_button.Enabled = false;
                s4_button.Enabled = false;
                s5_button.Enabled = false;
                home_button.Enabled = false;
                brakes_button.Enabled = false;
                update_sensors_button.Enabled = false;
                rp_button.Enabled = false;
                ep_button.Enabled = false;
                rd_button.Enabled = false;
                ru_button.Enabled = false;
                error_rst.Enabled = false;
                MCM_CB.Enabled = false;
                serial_indicator.Text = "Not Connected";
                serial_indicator.BackColor = Color.Red;
                ListEvents.Items.Add(DateTime.Now.ToString() + ".....Serial
Port Closed");

            }
            else
            {
                try
                {
```

```
                    serialPort1.PortName = comboBox1.Text;
                    serialPort1.Open();
                }
                catch (Exception err)
                {
                    MessageBox.Show(err.Message, "Error",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    ListEvents.Items.Add(DateTime.Now.ToString() +
".....Serial Port Error:");
                    ListEvents.Items.Add(DateTime.Now.ToString() + "....." +
err.Message);
                }
                if (serialPort1.IsOpen)
                {
                    CDC.Text = ("Disconnect");
                    s1_button.Enabled = true;
                    s2_button.Enabled = true;
                    s3_button.Enabled = true;
                    s4_button.Enabled = true;
                    s5_button.Enabled = true;
                    home_button.Enabled = true;
                    brakes_button.Enabled = true;
                    update_sensors_button.Enabled = true;
                    rp_button.Enabled = true;
                    ep_button.Enabled = true;
                    rd_button.Enabled = true;
                    ru_button.Enabled = true;
                    error_rst.Enabled = true;
                    MCM_CB.Enabled = true;
                    serial_indicator.Text = "Connected";
                    serial_indicator.BackColor = Color.Green;
                    ListEvents.Items.Add(DateTime.Now.ToString() +
".....Serial Port Opened");
                    serialPort1.WriteLine("#N");
                    sensors();
                    ListEvents.Items.Add(DateTime.Now.ToString() +
".....Current  system status gathered from sensors");
                }
            }
        }

        private void label3_Click(object sender, EventArgs e)
        {

        }

        private void serialPort1_DataReceived(object sender,
SerialDataReceivedEventArgs e)
        {

        }

        private void s1_button_Click(object sender, EventArgs e)
        {
```

```
            s1_button.BackColor = Color.Magenta;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#a");
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Slide 1
Selected");
            sensors();
        }

        private void s2_button_Click(object sender, EventArgs e)
        {
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.Magenta;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#b");
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Slide 2
Selected");
            sensors();
        }

        private void s3_button_Click(object sender, EventArgs e)
        {
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.Magenta;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#d");
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Slide 3
Selected");
            sensors();
        }

        private void s4_button_Click(object sender, EventArgs e)
        {
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.Magenta;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#f");
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Slide 4
Selected");
            sensors();
        }

        private void s5_button_Click(object sender, EventArgs e)
        {
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
```

```
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.Magenta;
            serialPort1.WriteLine("#g");
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Slide 5
Selected");
            sensors();
        }

        private void home_button_Click(object sender, EventArgs e)
        {
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#E");
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Homing
function selected");
            sensors();
        }

        private void rd_button_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Realod Down
function selected");
            DialogResult rd_warning = MessageBox.Show("Please ensure the
bottom panel and bottom front bracket are removed before continuing.",
               "Reload Warning", MessageBoxButtons.OKCancel,
MessageBoxIcon.Warning);
            if (rd_warning == DialogResult.OK)
            {
                ListEvents.Items.Add(DateTime.Now.ToString() + ".....Reload
Down warning message OK selected");
                s1_button.BackColor = Color.MediumTurquoise;
                s2_button.BackColor = Color.MediumTurquoise;
                s3_button.BackColor = Color.MediumTurquoise;
                s4_button.BackColor = Color.MediumTurquoise;
                s5_button.BackColor = Color.MediumTurquoise;
                serialPort1.WriteLine("#U");
                sensors();
            }
        }

        private void ep_button_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Extend Pusher
function selected");
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#C");
```

```
            sensors();
        }

        private void brakes_button_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Brakes OFF
function selected");
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#T");
        }

        private void ru_button_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Reload Up
function selected");
            DialogResult ru_warning = MessageBox.Show("Please ensure you are
ready to reinsert the stack (Reload Up).",
                "Reload Warning", MessageBoxButtons.OKCancel,
MessageBoxIcon.Warning);
            if (ru_warning == DialogResult.OK)
            {
                ListEvents.Items.Add(DateTime.Now.ToString() + ".....Reload up
warning message OK selected");
                s1_button.BackColor = Color.MediumTurquoise;
                s2_button.BackColor = Color.MediumTurquoise;
                s3_button.BackColor = Color.MediumTurquoise;
                s4_button.BackColor = Color.MediumTurquoise;
                s5_button.BackColor = Color.MediumTurquoise;
                serialPort1.WriteLine("#S");
                sensors();
            }
        }

        private void rp_button_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Retract
Pusher function selected");
            s1_button.BackColor = Color.MediumTurquoise;
            s2_button.BackColor = Color.MediumTurquoise;
            s3_button.BackColor = Color.MediumTurquoise;
            s4_button.BackColor = Color.MediumTurquoise;
            s5_button.BackColor = Color.MediumTurquoise;
            serialPort1.WriteLine("#H");
            sensors();
        }

        private void update_sensors_button_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Update
sensors function selected");
```

```
            serialPort1.WriteLine("#N");
              sensors();

        }

        private void Form1_Load(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Event Logger
Started");
            ListEvents.Items.Add("----------------------------------------
--------------------");
        }



        private void tb_test_TextChanged(object sender, EventArgs e)
        {

        }

        private void label1_Click_1(object sender, EventArgs e)
        {

        }



        public void sensors()
        {
            int cont_read = 1;
            int pusher_status = 0;
            s1_button.Enabled = false;
            s2_button.Enabled = false;
            s3_button.Enabled = false;
            s4_button.Enabled = false;
            s5_button.Enabled = false;
            home_button.Enabled = false;
            brakes_button.Enabled = false;
            update_sensors_button.Enabled = false;
            rp_button.Enabled = false;
            ep_button.Enabled = false;
            rd_button.Enabled = false;
            ru_button.Enabled = false;
            error_rst.Enabled = false;

            indicator.Text = ("Moving");
            indicator.BackColor = Color.Red;
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Device
Moving");
            while (cont_read == 1)
            {
                int j = 1;
                int indata_length;
                int star = 0;
```

```
int read_error = 0;
int i;
int x_position = 0;
int y_position = 0;
int error_code = 0;
while (serialPort1.BytesToRead == 0)
{ }
string indata = serialPort1.ReadLine();
char[] total_data = new char[100];
total_data = indata.ToCharArray();
char[] rfid_data = new char[100];
char[] pg_data = new char[100];
char[] Hall_data = new char[100];
char[] tms_data = new char[100];
char[] bms_data = new char[100];
char[] vert_data = new char[100];
char[] horiz_data = new char[100];
char[] error_data = new char[100];


while (star == 0  && read_error <= 25)
{
    if (total_data[0] == '*')
    {
        star = 1;
    }
    else
    {
        indata = serialPort1.ReadLine();
        serialPort1.DiscardInBuffer();
        total_data = indata.ToCharArray();
        star = 0;
        read_error = read_error + 1;
    }
}
   /* while (star == 0)
 {

     indata_length = indata.Length;

     while (total_data[j] != '*' && j <= indata_length)
     {
         j = j + 1;
     }
     if (j > indata_length)
     {
         j = j - 1;
     }
     if (total_data[j] == '*')
     {
         star = 1;
     }
     else
     {
```

```
            indata = serialPort1.ReadExisting();
            indata_length = indata.Length;
            total_data = indata.ToCharArray();
            j = 1;
        }
    }*/
    if (read_error < 25)
    {
        j = 1;
        //RFID data read
        i = 0;
        while (total_data[j] != '^')
        {
            rfid_data[i] = total_data[j];
            i = i + 1;
            j = j + 1;
        }
        string dataset1 = new string(rfid_data);
        rfid_sensor.Text = dataset1;
        //photogate data read
        j = j + 1;
        i = 0;
        while (total_data[j] != '^')
        {
            pg_data[i] = total_data[j];
            i = i + 1;
            j = j + 1;
        }
        string dataset2 = new string(pg_data);
        pg_sensor.Text = dataset2;
        //Hall effect data read
        j = j + 1;
        i = 0;
        while (total_data[j] != '^')
        {
            Hall_data[i] = total_data[j];
            i = i + 1;
            j = j + 1;
        }
        string dataset3 = new string(Hall_data);
        Hall_sensor.Text = dataset3;
        //Top micro switch data read
        j = j + 1;
        i = 0;
        while (total_data[j] != '^')
        {
            tms_data[i] = total_data[j];
            i = i + 1;
            j = j + 1;
        }
        string dataset4 = new string(tms_data);
        tms_sensor.Text = dataset4;
        //Bottom micro switch data read
        j = j + 1;
```

```
            i = 0;
            while (total_data[j] != '^')
            {
                bms_data[i] = total_data[j];
                i = i + 1;
                j = j + 1;
            }
            string dataset5 = new string(bms_data);
            bms_sensor.Text = dataset5;
            //Vertical Position data read
            j = j + 1;
            i = 0;
            while (total_data[j] != '^')
            {
                vert_data[i] = total_data[j];
                i = i + 1;
                j = j + 1;
            }
            string dataset6 = new string(vert_data);

            y_position = int.Parse(dataset6);
            y_prog.Value = y_position;
            //Horizontal position data read
            j = j + 1;
            i = 0;
            while (total_data[j] != '^')
            {
                horiz_data[i] = total_data[j];
                i = i + 1;
                j = j + 1;
            }
            string dataset7 = new string(horiz_data);
                //x_position = int.Parse(dataset7);
                //x_porg.Value = x_position;
                if (x_position == 100)
                 {
                    pusher_indicator.BackColor = Color.Green;
                    pusher_indicator.Text = "Inserted";
                 }
                else if (x_position == 0)
                 {
                     pusher_indicator.BackColor = Color.White;
                     pusher_indicator.Text = "Retracted";
                 }
                else
                 {
                     pusher_indicator.BackColor = Color.Red;
                     pusher_indicator.Text = "Moving";
                 }
        //Error data read
        j = j + 1;
            i = 0;
            while (total_data[j] != '!')
            {
```

```
                    error_data[i] = total_data[j];
                    i = i + 1;
                    j = j + 1;
                }
                string dataset8 = new string(error_data);
                error_code = int.Parse(dataset8);
                //End of sensor data
                //Check for end listen command
                j = j + 1;
                if (total_data[j] == '$')
                {
                    cont_read = 0;
                }

                pusher_status = int.Parse(dataset3);
                if (pusher_status < 100)
                {
                    pusher_indicator.BackColor = Color.Magenta;
                    pusher_indicator.Text = "Extended";
                }
                else
                {
                    pusher_indicator.BackColor = Color.White;
                    pusher_indicator.Text = "Retracted";
                }
            }
            if (error_code == 1)
            {
                ListEvents.Items.Add(DateTime.Now.ToString() + ".....Error
encountered: Error Code 1: Filter not removed");
                MessageBox.Show("The system has encountered an error.\n
Error Code 1\n Filter not removed.\n Recomendation: Manual retrival. Use
Manual Control to deactivate Pusher brakes.",
                                "System Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                error_indicator.BackColor = Color.Red;
                error_label.Text = "Code: 1";
            }
            else if (error_code == 2)
            {
                ListEvents.Items.Add(DateTime.Now.ToString() + ".....Error
encountered: Error Code 2: Filter not inserted");
                MessageBox.Show("The system has encountered an error.\n
Error Code 2\n Filter not inserted.\n Recomendation: Check for partial
insertion and manually retrive if necessary. Use Manual Control to deactivate
Pusher brakes.",
            "System Error", MessageBoxButtons.OK, MessageBoxIcon.Error);
                error_indicator.BackColor = Color.Red;
                error_label.Text = "Code: 2";
            }
            else if (error_code == 3)
            {
```

```
                    ListEvents.Items.Add(DateTime.Now.ToString() + ".....Error
encountered: Error Code 3: Selected filter not vertically alligned for
insertion");
                    MessageBox.Show("The system has encountered an error.\n
Error Code 3\n Selected filter not vertically alligned for insertion.\n Rehome
the device and try again.",
                                    "System Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    error_indicator.BackColor = Color.Red;
                    error_label.Text = "Code: 3";
                }
                else if (error_code == 4)
                {
                    ListEvents.Items.Add(DateTime.Now.ToString() + ".....Error
encountered: Error Code 4: Homing Failed");
                    MessageBox.Show("The system has encountered an error.\n
Error Code 4\n Homing failed.\n Check all conections inside the device.",
                                    "System Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    error_indicator.BackColor = Color.Red;
                    error_label.Text = "Code: 4";
                }
                else
                {
                    error_indicator.BackColor = Color.Green;
                    error_label.Text = "No error";
                }
            }



            s1_button.Enabled = true;
            s2_button.Enabled = true;
            s3_button.Enabled = true;
            s4_button.Enabled = true;
            s5_button.Enabled = true;
            home_button.Enabled = true;
            brakes_button.Enabled = true;
            update_sensors_button.Enabled = true;
            rp_button.Enabled = true;
            ep_button.Enabled = true;
            rd_button.Enabled = true;
            ru_button.Enabled = true;
            error_rst.Enabled = true;

            indicator.Text = ("Stationary");
            indicator.BackColor = Color.Green;
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Device
Stationary");

        }

        private void error_rst_Click(object sender, EventArgs e)
```

```
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Error reset
selected");
            serialPort1.WriteLine("#Z");
            sensors();
        }

        private void indicator_Click(object sender, EventArgs e)
        {

        }

        private void MCM_CB_CheckedChanged(object sender, EventArgs e)
        {
            while (MCM_CB.Checked == true)
            {
                s1_button.BackColor = Color.MediumTurquoise;
                s2_button.BackColor = Color.MediumTurquoise;
                s3_button.BackColor = Color.MediumTurquoise;
                s4_button.BackColor = Color.MediumTurquoise;
                s5_button.BackColor = Color.MediumTurquoise;
                ListEvents.Items.Add(DateTime.Now.ToString() + ".....Manual
Control Activated: Beginning sensor updates");
                sensors();
            }
            ListEvents.Items.Add(DateTime.Now.ToString() + ".....Manual
Control Deactivated: Ending sensor updates");
        }

        private void groupBox8_Enter(object sender, EventArgs e)
        {

        }

        private void save_log_Click(object sender, EventArgs e)
        {
            ListEvents.Items.Add(DateTime.Now.ToString() + @".....Event Log
Saved to C:\TEMP");
            string path = @"C:\TEMP\NAFS_Log_" + string.Format("{0:yyyy-MM-
dd_HH-mm-ss}", DateTime.Now) + ".txt";
            using (StreamWriter sw = File.CreateText(path))
            {
                foreach (string line in ListEvents.Items)
                    sw.WriteLine(line);
            }
        }

        private void pusher_indicator_Click(object sender, EventArgs e)
        {

        }
    }

}
```

Appendix E

CAD DRAWINGS

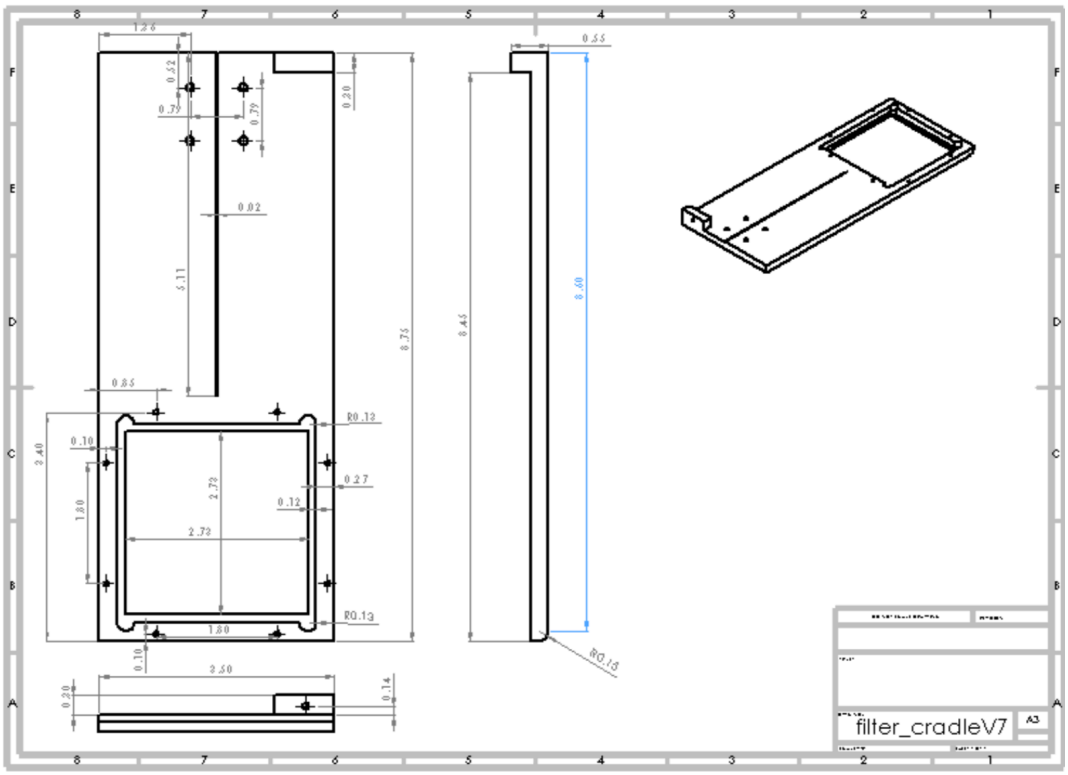

Figure E.1: Filter Slide

Appendix E: CAD Drawings

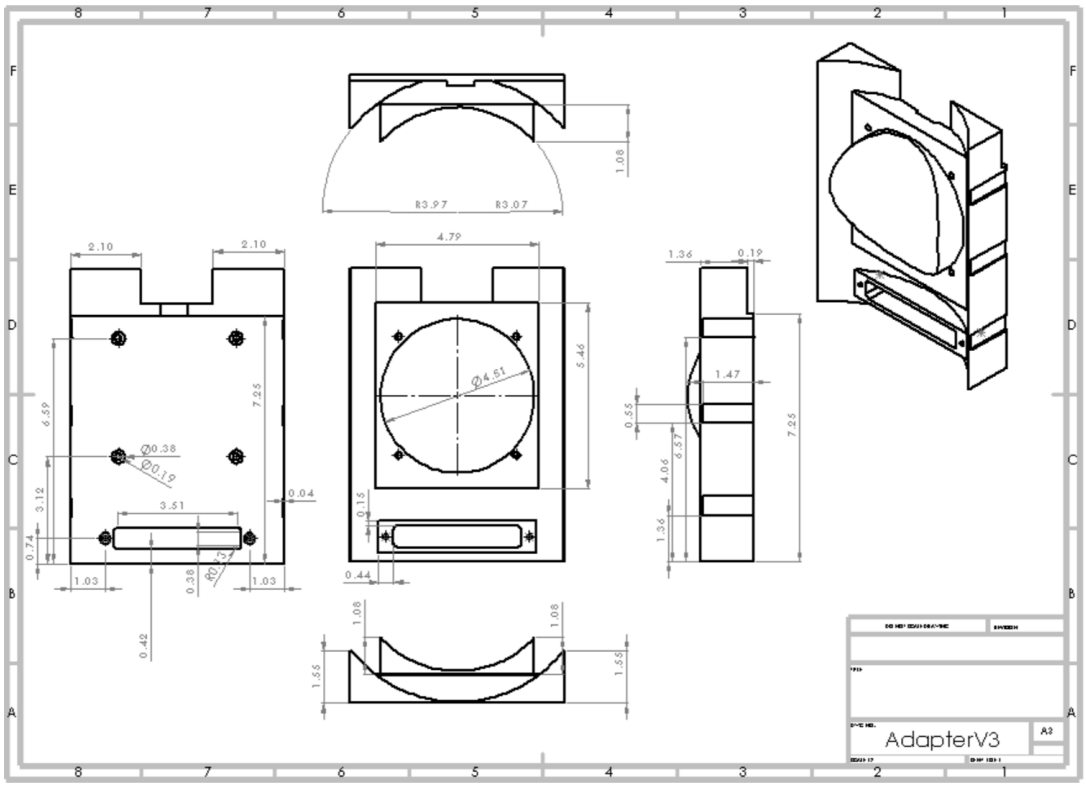

Figure E.2: Front Interface Mount

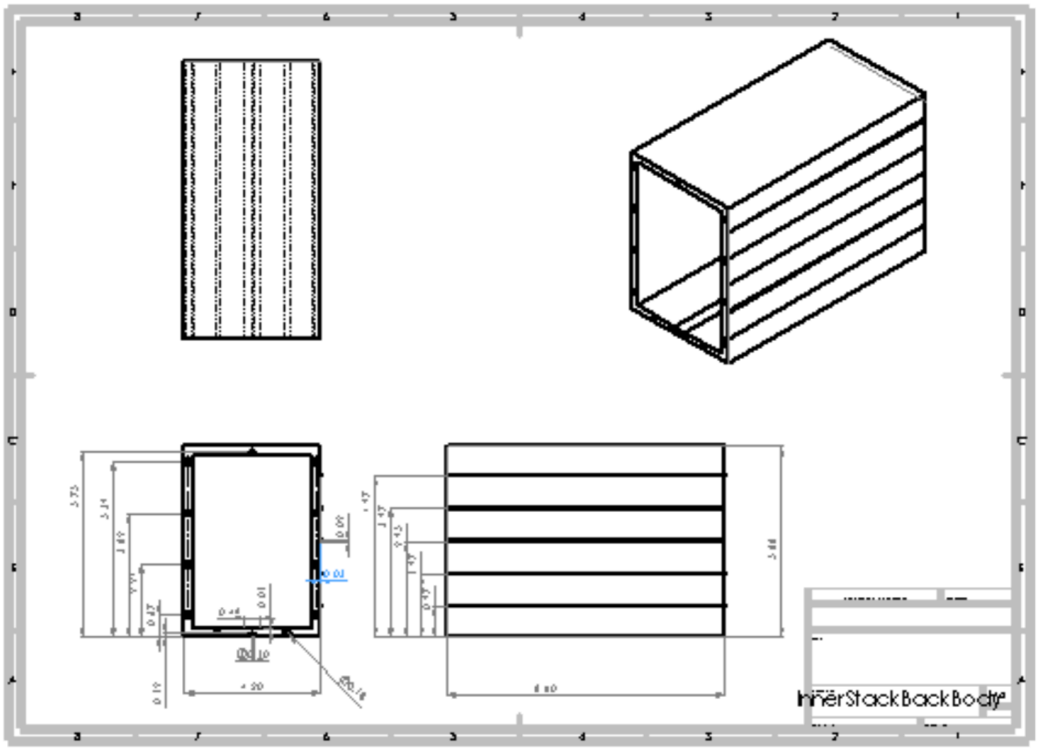

Figure E.3: Inner Stack main body

Appendix E: CAD Drawings

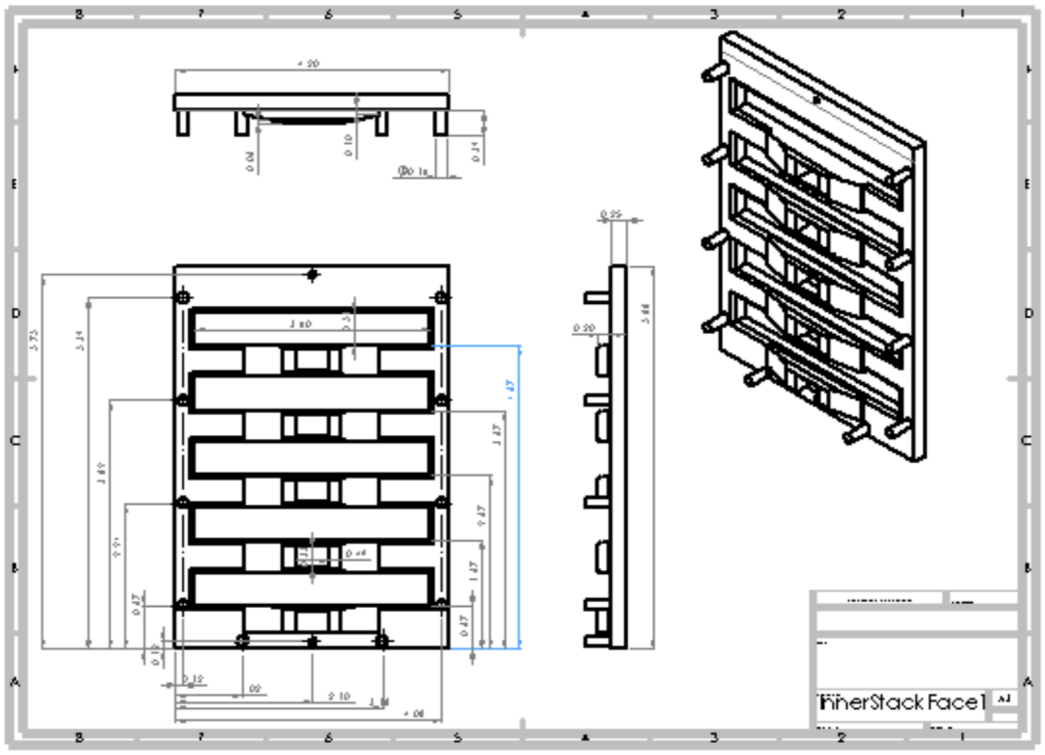

Figure E.4: Inner Stack front face

Appendix E: CAD Drawings

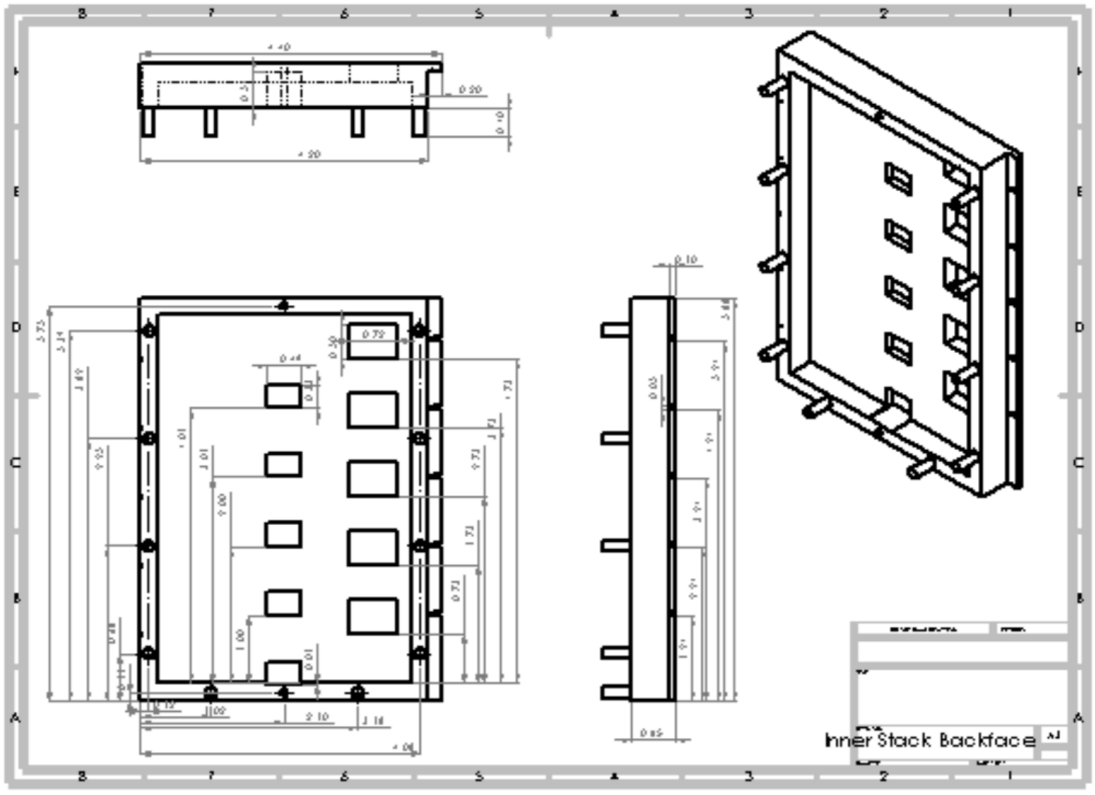

Figure E.5: Inner Stack rear face

Appendix F

ARDUINO (SoC) INPUTS/OUTPUTS

Table F.1

Arduino (SoC) I/O's		
PIN	**I/O**	**Description**
22	Output	Vertical motor pulses
23	Output	Horiz. Motor direction
24	Output	Vertical motor direction
25	Output	Horiz. Motor pulses
26	Output	Vertical motor enable
27	Output	Horiz. Motor enable
44	Input	Top microswitch value
45	Output	Relay control
46	Input	Bottom microswitch value
47	Output	Friction brake Relay control
A3	Input	MCU toggle switch value
A4	Output	LCD backlight control
A5	Input	MCU selection potentiometer value
A6	Input	Photogate sensor value
A7	Input	Hall effect sensor value